Fachgebiet Simulation und Systemoptimierung
Fachbereich Informatik
Technische Universität Darmstadt

ActiveVision – Intelligente Bildverarbeitung für den Einsatz im RoboCup

ActiveVision – Intelligent image processing for use in RoboCup

Diplomarbeit

von

Nicola Michael Gutberlet

Darmstadt, November 2007

Aufgabenstellung: Prof. Dr. Oskar von Stryk
Betreuer: Dipl.-Inform. Sebastian Petters

Ehrenwörtliche Erklärung

Hiermit versichere ich, die vorliegende Diplomarbeit ohne Hilfe Dritter und nur mit den angegebenen Quellen und Hilfsmitteln angefertigt zu haben. Alle Stellen, die aus den Quellen entnommen wurden, sind als solche kenntlich gemacht worden. Diese Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

Darmstadt, 21. November 2007

_____

Nicola Michael Gutberlet

Zusammenfassung

Das Fachgebiet Simulation und Systemoptimierung der Technischen Universität Darmstadt nimmt schon seit mehreren Jahren mit dem Team der Darmstadt Dribblers an der Weltmeisterschaft im Roboterfußball, dem *RoboCup*, teil. Trotzdem man bereits viele Erfolge in der Humanoiden-Liga hat sammeln können, muss und will man die Entwicklung stetig vorantreiben, um weiterhin konkurrenzfähig zu sein. Die Bildverarbeitung nimmt dabei eine sehr wichtige Schlüsselrolle ein, da sie einerseits die Umwelt des Roboters erfasst, und andererseits in der Regel einen großen Teil der zur Verfügung stehenden Ressourcen beansprucht.

In dieser Arbeit wird ein neuartiger Ansatz vorgestellt, der die Performanz der Bildverarbeitung signifikant steigern, Erkennungszuverlässigkeit und -robustheit aber mindestens beibehalten oder sogar steigern soll. Das Konzept, welches genau dieses verwirklichen soll, wird ActiveVision genannt. Es verarbeitet Bilder nicht nach dem immer gleichen Muster, sondern verwendet sogenannte Metadaten, die zum einen aus der Modellierung der Gesamtanwendung stammen und zum anderen auch aus Expertenwissen abgeleitet werden. Anhand der zugrunde liegenden Informationen können Objekte innerhalb des Bildes zielgerichteter und somit effizienter gefunden werden.

Abstract

The Simulation and Systems Optimizing Group of the Technische Universität Darmstadt participates with its team Darmstadt Dribblers in the world championship of robot soccer, called *RoboCup*, for several years. Although there exist many prosperities in the Humanoid-League for the Darmstadt Dribblers already, it is necessary and intended to keep development going to stay competitive. Image processing is an important key role, since it both detects the environment of the robot and consumes a lot of the available resources.

In this work a new approach will be introduced, that shall increase the performance of the image processing significantly without affecting its reliability and robustness of perception in a negative manner. The concept that is going to provide exactly these properties is named ActiveVision. It processes images no longer always the same way, but uses meta data that is derived from the entire application's modeling process as well as from some expertise. By means of this information objects will be recognized more goal-oriented and more efficient.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

1 Vorwort

1.1 Ziel der Arbeit

Ziel dieser Arbeit ist es, ein neues Konzept für die Bildverarbeitung auf einem humanoiden Roboter zu entwerfen, welches dem Einsatz im *RoboCup*, der Weltmeisterschaft im Roboterfußball, dienen soll.

Daraus ergibt sich eine genaue Spezifikation der Umgebung, in der das Bildverarbeitungssystem zum Einsatz kommt. Das bedeutet zum einen, dass das Auftreten von Objekten wohl definiert, zum anderen aber auch, dass die zur Verfügung stehende Rechenleistung stark eingeschränkt ist. Diese klaren Vorgaben bezüglich des Einsatzumfeldes sollen zur Steigerung der Performanz hinsichtlich der Ausführungsgeschwindigkeit der Bildverarbeitung genutzt werden. Dabei gilt es, die Erkennungsgenauigkeit von relevanten Objekten möglichst beizubehalten oder zu steigern.

Außerdem wird untersucht, inwiefern sogenannte Metadaten zur Verbesserung der Bildverarbeitung beitragen können. Als Metadaten seien einerseits Informationen aus der internen Modellierung von Umwelt und Objekten (zum Beispiel Modellierung der aktuell erkannten Ballposition und -bewegung) und andererseits das Wissen über die reale Umwelt und die realen Objekte (zum Beispiel Auftreten und Form von Feldlinien) bezeichnet. Zum Zweck der Untersuchung wurden die Metadaten in geeigneter Form bei der Umsetzung der Bildverarbeitung berücksichtigt.

Zusammenfassend lässt sich sagen, dass die Aufgabe der ActiveVision darin besteht, die Ausführungsgeschwindigkeit bei der Extraktion relevanter Informationen aus aufgenommenen Bildern dadurch zu steigern, dass anhand von Metainformationen entschieden werden kann,

- an welchen Stellen im Bild man anfängt nach Objekten zu suchen
- an welchen Stellen im Bild nicht (mehr) nach bestimmten Objekten gesucht werden muss
- ob eine Ansammlung von Bildpunkten als erkanntes Objekt bewertet werden kann, was zur Folge hat, dass nach diesem Objekt nicht weiter im Bild gesucht werden muss

Diese aktive Suche innerhalb von Bildern bedingt die Bezeichnung des neuen Konzepts als ActiveVision.

1.2 Aufbau der Arbeit

Zur besseren Übersicht soll an dieser Stelle kurz ein Überblick über die einzelnen Kapitel dieser Arbeit gegeben werden. Sie stellen zum einen die ActiveVision selbst dar, geben aber auch eine Übersicht über andere Konzepte. Abschließend werden erste Testergebnisse präsentiert und ein Ausblick auf weitere Entwicklungsmöglichkeiten gegeben.

Im Detail sieht die Gliederung wie folgt aus:

Kapitel 2:

Beweggründe für den Entwurf eines neuen Bildverarbeitungskonzepts.

Kapitel 3:

Präsentation anderer und ähnlicher Konzepte zur Bildverarbeitung im *RoboCup*.

Kapitel 4:

Anforderungen und Ziele, die an das Konzept ActiveVision geknüpft sind.

Kapitel 5:

Vorstellung des Konzepts der ActiveVision.

Kapitel 6:

Darstellung der konkreten Umsetzung.

Kapitel 7:

Erste Ergebnisse des neuen Ansatzes im Vergleich zum bisherigen.

Kapitel 8:

Zusammenfassung des Konzepts und Ausblick auf mögliche Erweiterungen.

2 Motivation

Die Bildverarbeitung ist ein wichtiger Faktor, der die Performanz des gesamten Robotersystems beeinflusst. Zum einen ist sie oft sehr rechenintensiv, zum anderen hängt die Genauigkeit der Steuerung maßgeblich von der Präzision und Erkennungsrate von Objekten in der Roboterumgebung ab.

Um Objekte zuverlässig zu erkennen, bestehen bereits etliche Standardverfahren in der Bildverarbeitung. Es gibt unterschiedliche Filter, die ein Bild zur Weiterverarbeitung vorbereiten können. Fehlende oder falsche Werte innerhalb des Bildes können ermittelt beziehungsweise korrigiert werden, Kanten können detektiert und darauf aufbauend ganze Merkmale extrahiert werden. Anhand detaillierter Merkmalsinformationen können dann ganze Objekte identifiziert werden.

Diese Verfahren besitzen in der Regel eine gute Erkennungsgenauigkeit, benötigen aber ein hohes Maß an Rechenaufwand, da jedes einzelne Pixel – oft sogar mehrmals – betrachtet werden muss. Außerdem benötigt die Suche nach einer optimalen Beschreibung von Merkmalen zusätzliche Ressourcen. Um durch eine Punktwolke zum Beispiel eine geeignete Gerade zu legen, kann diese mittels Hough-Transformation bestimmt werden. Allerdings bedarf es dazu eines Behälters, der quantisiert alle Möglichkeiten für Geraden im zweidimensionalen Raum beinhaltet. Sollen Geraden in umfangreicherem Maße unterschieden werden, verbraucht der Behälter eine beachtliche Menge an Speicher.

Unter normalen Umständen stellen die Anforderungen an die zugrunde liegende Hardware keine besondere Herausforderung dar. Einerseits sind heutige Rechnerarchitekturen, die Bildverarbeitungsaufgaben übernehmen, in der Regel mit ausreichenden Ressourcen ausgestattet. Zum anderen kann häufig eine etwas längere Ausführungsdauer verkraftet werden.

Für den Einsatz im *RoboCup* stehen nur begrenzte Ressourcen zur Verfügung und die Anwendungen müssen in Echtzeit ausführbar sein. Auch hier gibt es schon Standardansätze, die von der Mehrheit der Teams angewendet werden. Auf diese soll in Kapitel 3 näher eingegangen werden. Diese Ansätze haben aber weiterhin den Nachteil, mehr Rechenaufwand zu produzieren, als durch ein intelligenteres Konzept erreicht werden könnte.

Die Motivation dieser Arbeit liegt darin, ein Konzept zu entwerfen, dass die bisherigen Standardansätze speziell im *RoboCup* bezüglich ihrer Performanz deutlich über-

trifft und dabei Erkennungsrate und -genauigkeit beibehält oder sogar verbessert. Dieses soll erreicht werden, indem anhand von Metadaten (siehe Abschnitt 1.1) Entscheidungen über die Verarbeitung des aktuellen Bildes getroffen werden. Unter anderem kann somit, schon bevor das komplette Bild analysiert wurde, die Ballerkennung abgeschlossen werden. Das hat zur Folge, dass nicht jede Iteration der Bildverarbeitung das komplette Bild verarbeiten muss.

Als Vergleich und zur Bewertung des neuen Bildverarbeitungsmoduls dient der bisherige Ansatz der Darmstadt Dribblers, der schon erfolgreich im *RoboCup* eingesetzt wurde. Auch dieser wird in Kapitel 3 genauer beschrieben.

3 Stand der Forschung

3.1 Standardverfahren zur Bildverarbeitung im *RoboCup*

Dieser Abschnitt beschreibt, welche Standardverfahren zur Bildverarbeitung unter den Teams im *RoboCup* weit verbreitet sind. Viele Ansätze basieren auf der Erstellung eines Rasters über das gesamte zu verarbeitende Bild (siehe Abschnitt 3.1.2) mit anschließendem Zusammenfassen von Farbpixeln zu sogenannten *Blobs* (siehe Abschnitt 3.1.3). Außerdem wird häufig eine Farbtabelle verwendet (siehe Abschnitt 3.1.1).

Auch die derzeitig eingesetzte Bildverarbeitung der Darmstadt Dribblers basiert auf diesen Standardkonzepten und wird in Abschnitt 3.4 genauer beschrieben.

3.1.1 Verwendung einer Farbtabelle

Farbtabellen werden in der Regel offline vor der eigentlichen Ausführung der Anwendung erzeugt. Sie stellen eine Zuordnung von rohen Farbwerten zu Farbklassen dar. Dadurch können während der Bildverarbeitung Farbpixel effizient klassifiziert werden. Ein Nachteil ist jedoch der nicht unerhebliche Speicherbedarf einer solchen Farbtabelle. Um einen effizienten Zugriff gewährleisten zu können, muss sie für jedes mögliche Tripel an rohen Eingangswerten die zugehörige Farbklasse speichern.

In der aktuellen Bildverarbeitung der Darmstadt Dribblers und auch weiterhin im Ansatz der ActiveVision wird allerdings dennoch ebenfalls eine Farbtabelle eingesetzt und über ein dreidimensionales Array gespeichert. Hier wiegt der Effizienzgewinn den Speicherbedarf auf.

3.1.2 Aufspannen eines starren Rasters über das gesamte Bild

Das Aufspannen eines Rasters über das gesamte zu verarbeitende Bild dient hauptsächlich dazu, den Rechenaufwand der Bildverarbeitung dadurch zu verringern, dass

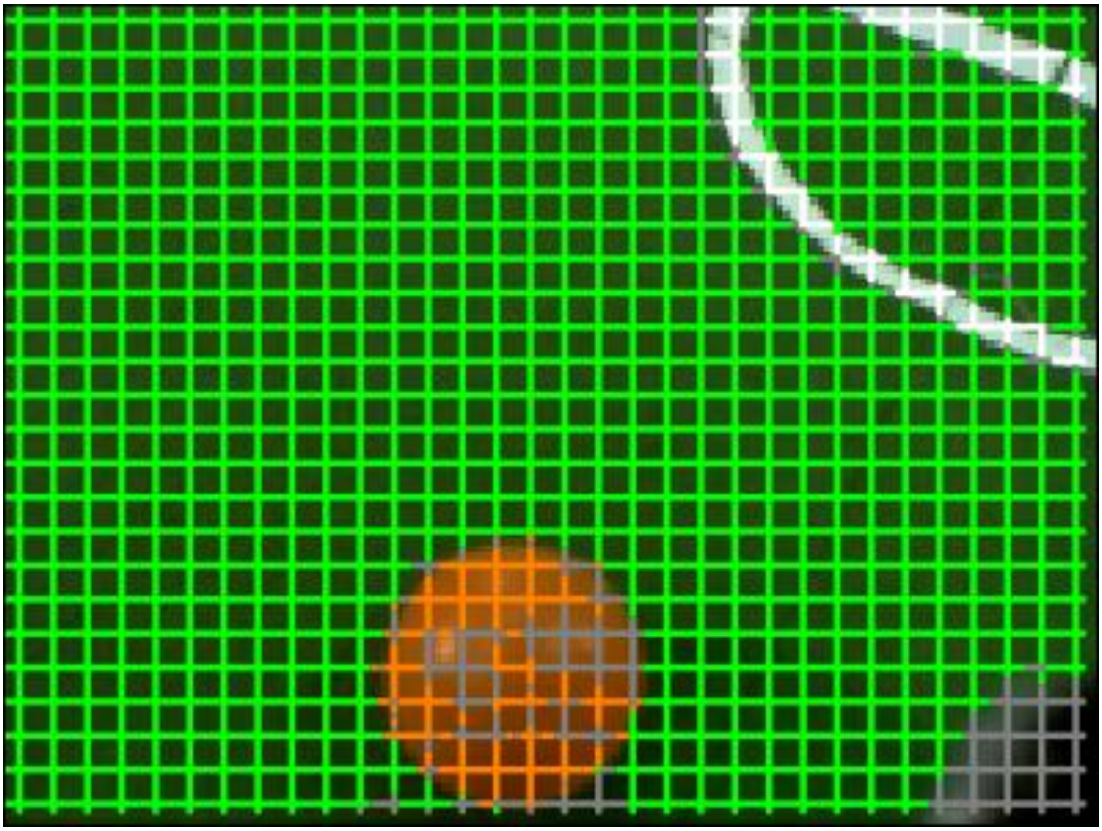


Abbildung 3.1: Beispiel eines starren Rasters (konstant fünf Pixel Abstand zwischen den Linien), das über ein zu verarbeitendes Bild gelegt wird. Hier aus dem bisherigen Ansatz der Darmstadt Dribblers. Jedes Pixel der horizontalen und vertikalen Gitterlinien wird klassifiziert.

nur eine Untermenge aller Bildpixel betrachtet wird. Hierzu wird im Vornhinein festgelegt nach welchem festen Muster Pixel betrachtet werden. Zum Beispiel könnte jede fünfte Zeile und Spalte des Bildes komplett betrachtet werden (siehe Abbildung 3.1). Oder man wählt lediglich einzelne Pixel, die einen festen Abstand zu einander haben, anstatt ganzer Zeilen und Spalten. Im ersten Fall müssten dann – zumindest in den ersten Schritten der Bildverarbeitung – lediglich circa 2/5 aller Pixel klassifiziert werden. In folgenden Schritten können dann noch weitere benötigte Pixel gezielt klassifiziert werden.

Der Vorteil dieses Vorgehens liegt darin, dass nur eine Untermenge aller Pixel verarbeitet werden muss, was die Ausführungszeit der Bildverarbeitung signifikant verringert. Allerdings müssen durch ein starres Raster auch Einschränkungen in Kauf genommen werden. Eventuell können Objekte, die im aktuellen Bild nur relativ klein erscheinen, „übersehen" werden. Außerdem verhindert ein unflexibles Suchmuster, dass die Suchstrategie an die aktuelle Situation angepasst werden kann. Wie noch gezeigt werden wird, verfolgt die ActiveVision einen solchen anpassungsfähigen Ansatz.

3.1.3 Erzeugen von Pixelregionen gleicher Farbe (*Blobs*)

Die Formierung von Farbregionen hat zum Ziel, Pixel in aussagekräftige Gruppen gleicher Farbe zusammenzufassen. Diese Gruppen werden *Blobs* genannt. Dieses ist in der Regel die aufwendigste Komponente der gesamten Bildverarbeitung. Im einzelnen läuft sie wie folgt ab.

Als erstes werden Pixel zu *RLE-Objekten* verknüpft. *RLE-Objekte* beschreiben Teile von Zeilen beziehungsweise Spalten nach dem Konzept des „Run-Length Encoding" (*RLE*). Das bedeutet, eine Sequenz von Pixeln gleicher Farbe innerhalb einer Zeile oder Spalte wird durch ihren Anfangspunkt und ihre Länge (möglich ist aber auch die Angabe des Endpunktes) beschrieben. Dabei können schon *RLE-Objekte* verworfen werden, die keine relevanten Informationen enthalten (zum Beispiel weil sie zu klein sind oder eine irrelevante Farbe repräsentieren).

Danach werden die nahe bei einander liegenden *RLE-Objekte* sukzessive zu den *Blobs*, also zu Regionen im Wesentlichen einer bestimmten Farbe, zusammengefasst. Diese *Blobs* sollten dann schon Daten enthalten, die die weitere Verarbeitung erleichtern. Als Beispiel hierfür wären Angaben über die Anzahl der enthaltenen Pixel zu nennen oder maximale und minimale Koordinaten des *Blobs* innerhalb des Bildes.

Anschließend werden die extrahierten Informationen den nachfolgenden Bildverarbeitungsprozessen zur Verfügung gestellt, welche dann daraus die relevanten Objekte identifizieren. Hierbei liegt der Vorteil darin, dass die folgenden Verarbeitungsschritte auf detaillierten aber zusammengefassten Daten arbeiten können. In welcher Form diese genau vorliegen, hängt von der Implementierung der folgenden Schritte ab. Für eine Torerkennung zum Beispiel könnten zwei Punkte ausreichend sein, die ein Rechteck definieren, das die entsprechende Farbregion einschließt.

Auch dieses Konzept findet in der aktuellen Bildverarbeitung der Darmstadt Dribblers Verwendung (siehe Abschnitt 3.4). Die ActiveVision geht schon bedingt durch das Grundkonzept einen etwas anderen Weg, wie in den Kapiteln 5 und 6 dargelegt werden wird.

3.2 Konventionelle Ansätze zur Bildverarbeitung im *RoboCup*

Dieser Abschnitt stellt kurz zwei Ansätze zur Bildverarbeitung vor, die eher als konventionell bezeichnet werden können. Sie verwenden im Wesentlichen die vorgestellten Standardverfahren (siehe Abschnitt 3.1) und sollen beispielhaft aufzeigen, inwiefern diese Standardverfahren bei andern Teams zum Einsatz kommen.

3.2.1 Konventioneller Ansatz der Universität von Texas in Austin[1]

Der Ansatz einer Bildverarbeitung der Universität von Texas basierend auf *Blobs* wird in „Real-Time Vision on a Mobile Robot Platform" [14] vorgestellt. Das Bildverarbeitungssystem lässt sich in vier Stufen unterteilen, die im Folgenden teilweise genauer beschrieben werden. Sie lauten:

1. Generierung einer Farbtabelle
2. Formierung von Farbregionen (*Blobs*)
3. Objekterkennung
4. Linienerkennung

Die ersten beiden Punkte werden an dieser Stelle nicht weiter ausgeführt, da sie allgemein in Abschnitt 3.1 beschrieben werden. Auch die Objekterkennung soll hier nur kurz beschrieben werden. Und zwar werden die erzeugten *Blobs* auf bestimmte Merkmale der zugeordneten Objekte überprüft, zum Beispiel ob die Ausmaße des *Blobs* gewisse Richtlinien erfüllen. Diese Merkmale sind an die Objekte angepasst. Das bedeutet beispielsweise, dass die Ausmaße eines Balls kleiner sein dürfen als die eines Tores. Die Klassifikationsrate wird mit den folgenden Werten angegeben: 100% bei stehendem, 92,7% bei laufendem Roboter. Außerdem sollen keine Objekte erkannt worden sein, die nicht auch im Bild vorhanden sind (False-Positives).

Abschließend soll die Linienerkennung etwas genauer beleuchtet werden, da diese sich nicht so direkt aus den Standardverfahren konstruieren lässt. Sie ist durch den in [10] von Jüngel und Röfer vorgestellten Ansatz motiviert. Im Detail läuft sie wie folgt ab.

Im ersten Schritt werden ausschließlich vertikale *Scan-Lines* (das heißt Spalten des Bildes mit klassifizierten Pixeln) festen Abstands erstellt. Diese werden dann von unten nach oben auf Kandidaten für Kanten von Feldlinien überprüft. Als solche Kandidaten werden nur Übergänge von der Hintergrundfarbe (Grün) zur Feldlinienfarbe (Weiß) – oder umgekehrt – akzeptiert, die zwei Kriterien erfüllen. Eine bestimmte Anzahl grüner Pixel muss unterhalb dieses Übergangs liegen, und der projizierte Punkt der Kante darf eine gewisse Entfernung zum Roboter nicht überschreiten. Nach dem ersten Fund eines passenden Übergangs innerhalb einer Scan-Line wird er gespeichert und diese Bildspalte nicht weiter betrachtet.

Die Kantenpixel werden dann mittels der Methode der kleinsten Fehlerquadrate [16] sukzessive zu Linien zusammengefasst. Dabei findet eine Rauschunterdrückung statt, indem Ausreißerkandidaten direkt verworfen werden. Außerdem werden berechnete

[1] URL: `http://www.cs.utexas.edu/`

Linien verworfen, die nicht eine bestimmte Mindestanzahl an Kantenpixeln beinhalten. Anschließend werden aus den gefundenen Feldlinien Kreuzungen bestimmt, welche für die Lokalisierung des Roboters eine höhere Aussagekräftigkeit als bloße Linien besitzen.

Auch bei der Linienerkennung wird eine hohe Klassifikationsrate (100% stehend, 93,3% laufend) und wieder 0% False-Positives erreicht.

3.2.2 Ansatz der Carnegie Mellon Universität[2] zur *Blob*-Generierung

Die unteren Ebenen einer Bildverarbeitung werden in „Fast and inexpensive color image segmentation for interactive robots" von Balch, Bruce und Veloso [2] beschrieben. Diese Low-Level-Bildverarbeitung wurde bereits in der Four-Legged und in der Small-Size League eingesetzt und wird im Folgenden genauer beschrieben.

Anfangs kann der Farbraum bei Bedarf in ein anderes Modelle überführt werden. Liegen die Pixelwerte also im gewünschten Format vor, werden alle Pixel klassifiziert. Dieses geschieht über drei einfache bool'sche Arrays, deren Einträge UND-verknüpft werden. Jede Farbklasse besitzt ihre eigenen drei Arrays, die eine Aussage darüber treffen, ob ein roher Farbwert zu dieser Klasse gehört. So ergibt sich die Zugehörigkeit eines Pixels zu einer Farbklasse wie folgt:

$$belongsToClass(C, y, u, v) = \begin{cases} C.classY[y] \\ \textbf{AND} \\ C.classU[u] \\ \textbf{AND} \\ C.classV[v] \end{cases}$$

Dabei stellt C die aktuell getestete Farbklasse, y, u und v die rohen Farbwerte des Pixels und $classX[x]$ die in den angesprochenen bool'schen Arrays gespeicherten Werte dar.

Der Aufwand dieser Farbklassifikation ist linear mit der Anzahl der Pixel und der Farbraumdimension. Als Vorteil kann gewertet werden, dass einzelne Pixel mehreren Farbklassen zugeordnet werden können. Das führt dazu, dass auch bezüglich einer eindeutigen Klassifikation kritische Pixel nicht für die späteren Bildverarbeitungsvorgänge verloren gehen.

Nach der Farbklassifikation werden für jede Zeile horizontale *RLE-Objekte* (auch als *Runs* bezeichnet) erzeugt und gleichfarbige zu Regionen zusammengefasst. Die-

[2]URL: `http://www.cs.cmu.edu/`

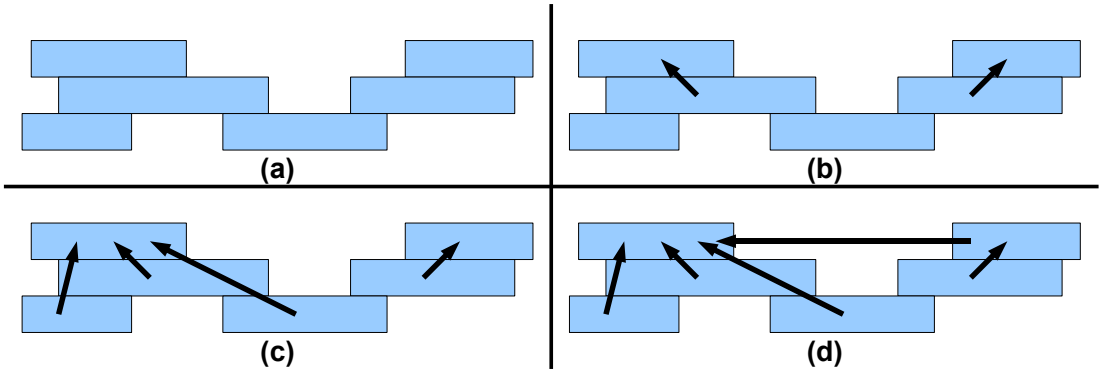


Abbildung 3.2: Zusammenfassen von *Runs* zu *Blobs* beim Ansatz der Carnegie Mellon Universität. Bild (a) stellt die Ausgangssituation überlappender *Runs* dar. In (b) ist zu sehen, wie *Runs* direkten Elternknoten zugeordnet werden. (c) zeigt, dass im weiteren Verlauf immer die obersten Vorgänger neuen *Runs* zugeteilt werden. Abschließend stellt (d) den resultierenden Baum eines *Blobs* dar.

ses geschieht mittels Union-Find-Algorithmus, der in der Praxis eine lineare Laufzeit aufweist, und läuft folgendermaßen ab:

1. *Runs* werden einander zugeordnet,
   - wenn sie auf zu einander benachbarten Zeilen liegen und
   - wenn sie die gleiche Farbe repräsentieren und
   - wenn sie sich horizontal gesehen überlappen

   Dadurch entsteht ein Wald, in dem jeder Knoten (= *Run*) auf seinen Elternknoten zeigt (von dem aus er erreicht wurde)
2. In einem zweiten Durchlauf über alle *Runs* wird für jeden Knoten die eindeutige Zugehörigkeit zu einer Farbregion bestimmt.

Die Entstehung eines einzelnen *Blobs* als Baum zeigt Abbildung 3.2.

Nach der Generierung von Farbregionen werden Statistiken für eben diese bestimmt, welche folgende Informationen beinhalten:

- Rechteck, das die Farbregion einschließen (*Bounding-Box*)
- Schwerpunkt der *Bounding-Box*
- Größe der *Bounding-Box*
- Dichte der *Bounding-Box* (Anteil der Pixel mit Region zugeordneter Farbe innerhalb der *Bounding-Box*)

Danach werden wiederum einzelne Farbregionen zu größeren zusammengefasst, wenn deren gemeinsame *Bounding-Box* eine gewisse Dichte noch erfüllt. Diese Regionen werden abschließend nach Farbe und Größe geordnet, um sie in späteren Schritten gezielter weiterverarbeiten zu können. Damit ist die Generierung von Farbregionen abgeschlossen.

3.3 Neue innovative Ansätze zur Bildverarbeitung im *RoboCup*

In diesem Abschnitt werden Ansätze anderer Teams zur Bildverarbeitung vorgestellt, wie sie für den *RoboCup* entwickelt und teilweise auch schon eingesetzt worden sind. Hierbei liegt der Fokus auf Ansätzen, die sich mit neuen Konzepten beschäftigen. Von den vorhandenen Konzepten werden insbesondere solche betrachtet, die eine gewisse Affinität zum Konzept der ActiveVision aufweisen. Außerdem kann an dieser Stelle jeweils nur ein Teil des jeweiligen Systems beschrieben werden.

3.3.1 Ansatz der Sharif-Universität in Teheran[3] zur Verwendung eines angepassten Suchrasters

[4] beschreibt einen Ansatz der Sharif-Universität, der besonders bezüglich der Auswahl von betrachteten Suchpunkten[4] innerhalb eines Bildes sehr interessant ist. Laut der Beschreibung zur Teilnahme am *RoboCup* 2001 [5] kam dieses Konzept hier in der Middle-Size League zum Einsatz.

Die Punkte im Bild, die hinsichtlich des Vorkommens von relevanten Objekten analysiert werden, werden wie folgt gesetzt. Der Abstand zwischen je zwei benachbarten Suchpunkten wird so gewählt, dass, wenn man diese auf den Boden projiziert, diese immer einen konstanten Abstand aufweisen (siehe Abbildung 3.3). Dieser Abstand soll gewährleisten, dass ein Rechteck, das das relevante Objekt mit den kleinsten Ausmaßen (in diesem Fall den Ball) einschließt, von mehreren Suchpunkten getroffen wird. Zu erwähnen wäre hierbei noch, dass die Kamera unbeweglich am Roboter angebracht ist, wodurch die Suchpunkte für jedes Bild gleich angeordnet sein können.

Die Suche nach Objekten selbst findet folgendermaßen statt. Der initiale Suchpunkt und die Reihenfolge in der weitere Punkte traversiert werden ist immer gleich. Angefangen wird in der untersten Spalte des Bildes, um dann nach und nach die Punkte der darüber liegenden Spalten zu analysieren.

[3] URL: `http://www.sharif.ir/en/`

[4] Als Suchpunkte werden in dieser Arbeit Punkte innerhalb des Bildes bezeichnet, die explizit bei der Suche nach Objekten analysiert werden.

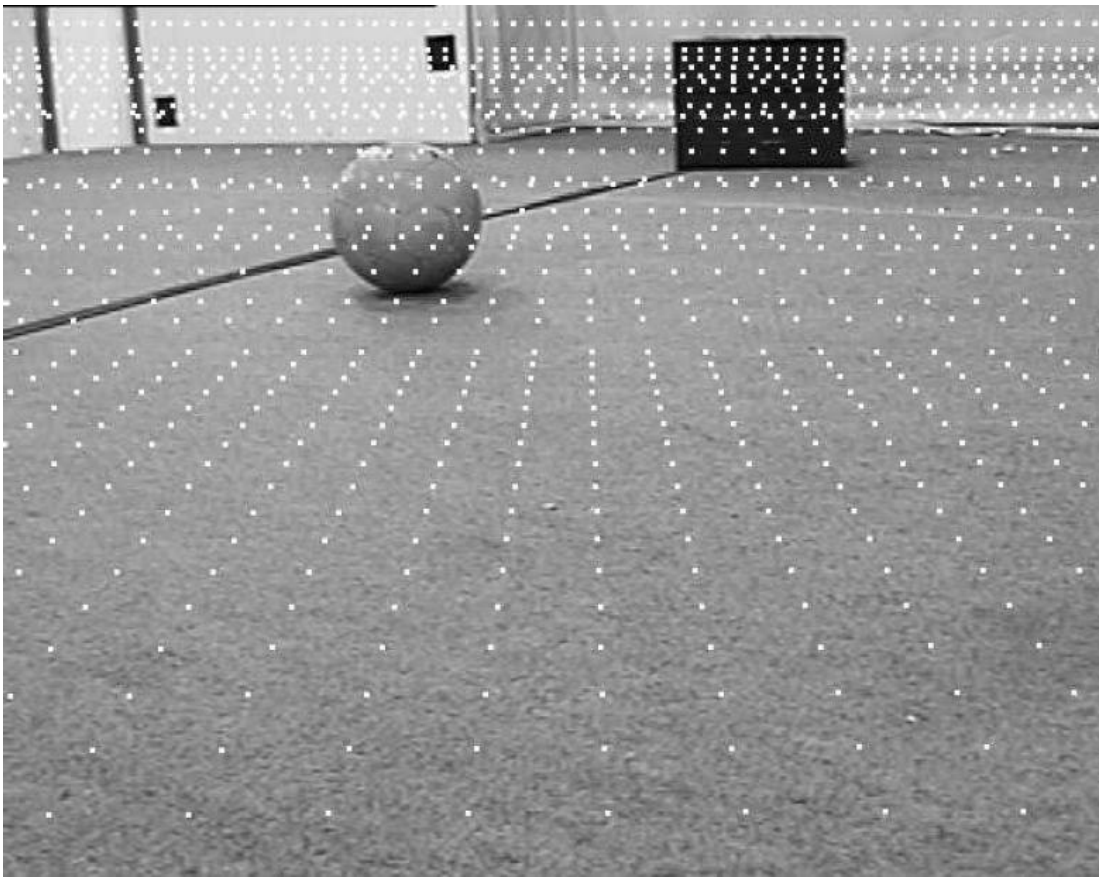


Abbildung 3.3: Suchpunkte beim Ansatz der Sharif-Universität (Quelle: [4]).

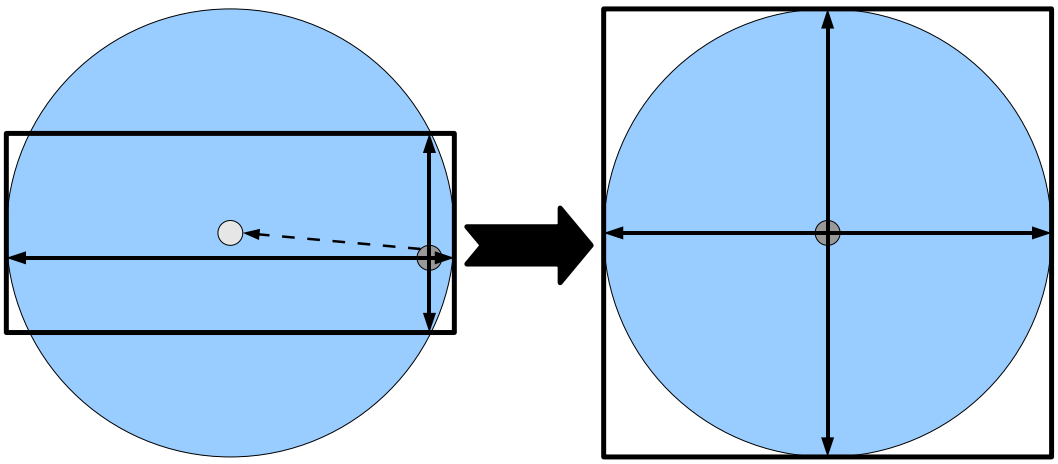


Abbildung 3.4: Bestimmung einer *Bounding-Box* für den Ball beim Ansatz der Sharif-Universität.

Wird nun bei der Analyse eines Pixels die einem Objekt zugeordnete Farbe identifiziert, kommt direkt der Objekt-spezifische Erkenner zur Ausführung. So sind nach einem kompletten Durchlauf aller Suchpunkte alle im Bild vorhandenen Objekte erkannt.

Abschließend soll beispielhaft noch kurz die Vorgehensweise des Ball-Erkenners geschildert werden. Ausgangspunkt ist das Pixel, dass die Ausführung des Ball-Erkenners ausgelöst hat, also die entsprechende Farbe darstellt. Im ersten Schritt wird von diesem ausgehend das Ende der Ball-spezifischen Farbe nach oben, unten, links und rechts gesucht. Dabei findet eine Rauschunterdrückung statt, indem ein Pixel anderer Farbe als Ball-farben angenommen wird, wenn eine bestimmte Anzahl seiner Nachbarpixel Ball-farben ist. Nach der Definition eines Rechteckes durch die gefundenen Ballränder wird der Abstand des Ausgangspunktes zum Mittelpunkt des Rechteckes berechnet. Überschreitet dieser einen kritischen Wert, was bedeutet, der Ausgangspunkt liegt relativ nahe am Ballrand, wird ausgehend vom Rechtecksmittelpunkt erneut nach den Ballrändern gesucht (siehe Abbildung 3.4). Diese Prozedur wird wiederholt, bis Ausgangs- und Mittelpunkt ähnlich genug sind. Anschließend wird das resultierende Rechteck einer Zuverlässigkeitsprüfung unterzogen, bei der kleine Rechtecke als Rauschen verworfen werden. Danach wird die Suche nach Objekten am nächsten Suchpunkt fortgesetzt.

3.3.2 Ansatz der Humboldt Universität Berlin[5] mit einem flexiblen Suchraster.

Bach und Jüngel beschreiben in „Using pattern matching on a flexible, horizon-aligned grid for robotic vision" [1] einen Ansatz, der bei der Suche ein Raster verwendet, dass sich an die aktuelle Kameraposition anpasst. Dieses Verfahren wurde für den Einsatz in der Four-Legged League entwickelt und beinhaltet keine Linienerkennung.

Das der Suche zugrunde liegende Raster ist wie folgt aufgebaut. Vertikale Rasterlinien werden perspektivisch gezogen. Das heißt, sie beginnen in einem Fluchtpunkt, dem Mittelpunkt des Horizontes innerhalb des aktuellen Bildes, und enden am unteren Rand des Bildes. Hier wird ein Abstand der Linien gewählt, der an die Größe des aktuell gesuchten Objektes (zum Beispiel des Balls) angepasst ist. Sie werden in späteren Schritten von unten nach oben verarbeitet.

Die horizontalen Gitterlinien verlaufen parallel zum berechneten Horizont. Auch ihr Abstand wird an die Größe des gesuchten Objektes angepasst. Die Berechnung der Abstände soll beispielhaft für die Erkennung des Balls verdeutlicht werden. Der Mittelpunkt des Horizonts wird auf den Boden projiziert. An der Stelle, an der der zugehörige Sichtstrahl zum Boden den Abstand des Balldurchmessers hat, wird ein neuer Punkt auf den Boden gesetzt. Dieser wird in die Bildebene der Kamera projiziert und dort als Mittelpunkt der nächsten horizontalen Rasterlinie gesetzt. Für diesen und die folgenden Sichtstrahlen wird wie mit dem des Horizontmittelpunktes verfahren. Anschließend wird der Abstand zweier Gitterlinien mindestens halbiert, um die Wahrscheinlichkeit zu mindern, ein Objekt zu „übersehen". Die Generierung des Rasters und das Endergebnis zeigt Abbildung 3.5.

Die Linien des Rasters werden dann auf bestimmte Muster überprüft. Zum Beispiel signalisiert eine Folge „Grün, Grün, Gelb, Gelb" den Rand des gelben Tores. Wird ein solches Muster erkannt, kommt es zur Ausführung des entsprechendes Erkenners an dieser Stelle.

Der Erkenner testet, ob der aktuelle Startpunkt bereits einem Objekt der gleichen Klasse zugeordnet wurde. Falls das nicht der Fall ist, bestimmt der Erkenner die Ausmaße des zugehörigen Objektes und speichert diese ab. Zum Schluss werden die erkannten Objekte auf ihre Konsistenz überprüft. So wird zum Beispiel aus mehrfach erkannten Objekten bestimmt, welches das reale repräsentiert.

[5] URL: `http://www.hu-berlin.de/`

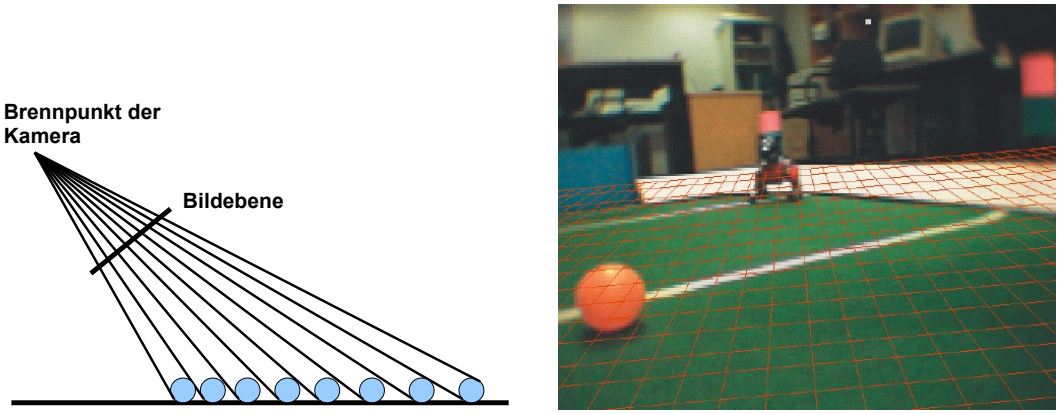


Abbildung 3.5: Generierung des flexiblen Rasters des Ansatzes der Humboldt Universität Berlin und beispielhaft das über ein reales Bild gelegte Resultat (Quelle: [1]).

3.3.3 Ansatz der Universität von Texas in Austin[6] zur gezielten Suche im Bild

Daniel Stronger und Peter Stone beschreiben in „Selective Visual Attention for Object Detection on a Legged Robot" [15] einen Ansatz, um in Bildern gezielt nach relevanten Objekten zu suchen. Der Ausdruck „Selective Attention" kann hierbei so verstanden werden, dass zuerst die Teile des Bildes verarbeitet werden, in denen relevante Objekte erwartet werden. Entwickelt wurde dieser Ansatz zum Einsatz in der Four-Legged League.

Erwartete Positionen von Objekten werden aus der zuletzt wahrgenommenen Position und der Bewegungsmodellierung des Objektes berechnet. Das bedeutet insbesondere, dass nicht der naive Ansatz gewählt wurde, die nächste Suche nach einem Objekt bei der zuletzt wahrgenommenen Position im Bild zu starten, sondern dass dieser Anfangspunkt durchaus an ganz anderer Stelle innerhalb oder sogar außerhalb des Bildes liegen kann. Das entspricht der Herausforderung, in einer teilweise hoch dynamischen Umwelt zu agieren.

Allerdings wurde dieser Ansatz bisher lediglich auf unbewegte Gegenstände wie Tore oder Landmarken angewendet. Trotzdem wurde wohl die Notwendigkeit einer Bewegungsmodellierung erkannt, da auch statische Objekte bezüglich des Roboter-eigenen Koordinatensystems ihre Positionen verändern.

Somit ergeben sich folgende Hauptherausforderungen:

1. Die Position des gesuchten Objektes innerhalb des Bildes muss vorhergesagt

[6] URL: `http://www.cs.utexas.edu/`

werden.

2. Eine Strategie muss definiert werden, mit der weiter nach einem Objekt gesucht wird, wenn es sich nicht an der vorhergesagten Position befindet.

3. Ein Verfahren muss entwickelt werden, wie die Erkennung eines Objektes abgeschlossen werden kann, wenn ein Pixel der Objekt-spezifischen Farbe gefunden wurde. Das heißt, es gilt festzulegen, welche weiteren Pixel im konkreten Fall untersucht werden müssen.

Der Abstand zwischen den einzelnen Suchpunkten bleibt innerhalb des Bildes konstant, wird allerdings pro Bild angepasst. Das bedeutet, der Abstand orientiert sich an den kleinsten Ausmaßen, den das gesuchte Objekt im aktuellen Bild annehmen kann. Ist zum Beispiel ein Tor innerhalb des aktuellen Bildes mindestens 50 Pixel hoch (das lässt sich unter anderem durch die Stellung der Kamera bestimmen), so wird der Abstand zwar kleiner als 50 Pixel gewählt, kann aber dennoch wesentlich größer sein, als wenn keine Angabe über die Torausmaße bestünde. Dann müsste nämlich auch der Fall berücksichtigt werden, in dem ein Tor beispielsweise nur 5 Pixel hoch ist.

Das gesamte Vorgehen des vorgestellten Bildverarbeitungssystems stellt sich dann wie folgt dar. Die Suche für jedes einzelne Objekt findet getrennt von den anderen statt. Als erstes wird genau eine Position berechnet, an der das gesuchte Objekt im Bild erwartet wird. Befindet sich diese Position außerhalb des Bildes, so wird die Suche nach diesem Objekt erst gar nicht gestartet. Im anderen Fall wird die folgende Suchstrategie solange angewendet, bis entweder das Bild komplett durchsucht oder das Objekt gefunden wurde. Analysiere, ob das Pixel an der aktuellen Suchposition eine Farbe des gesuchten Objektes aufweist. Kommt die Analyse zu einem negativen Ergebnis, wird die nächste Suchposition betrachtet. Die Reihenfolge der untersuchten Positionen verdeutlicht Abbildung 3.6.

Falls ein Pixel der Objekt-spezifischen Farben gefunden wird, werden Ausmaße und Lage des Objektes berechnet. Dazu wird, ausgehend von der aktuellen Position, der linke und rechte Farbrand innerhalb der Zeile bestimmt. Dabei werden vereinzelte Pixel anderer Farbe als Rauschen übersprungen. Dieser Schritt wird dann für die vorherigen und nächsten Zeilen wiederholt, bis der obere beziehungsweise untere Rand der Farbe erreicht wird.

Abschließend kann das Objekt als Rechteck bestimmt werden, welches durch die erste und letzte Zeile beziehungsweise Spalte mit der entsprechenden Farbe definiert wird (zur Erinnerung: der Ansatz behandelt lediglich rechteckige Objekte).

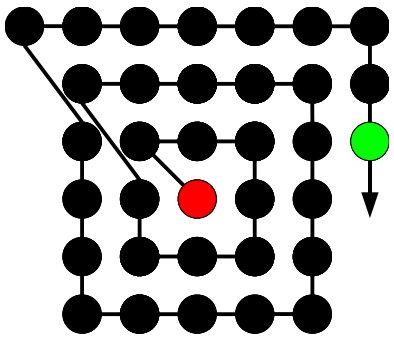


Abbildung 3.6: Reihenfolge der analysierten Pixel beim Ansatz der Universität von Texas.

3.3.4 Ansatz der Freien Universität Berlin[7] zur Verfolgung von Blobs

In „Tracking Regions" von von Hundelshausen und Rojas [9] wird ein Ansatz beschrieben, der sich damit befasst, nach Möglichkeit von jedem neu aufgenommenen Bild nur Teile verarbeiten zu müssen. Dieses geschieht, indem die relevanten identifizierten Farbregionen von Bild zu Bild verfolgt werden. Weitere Informationen und die Integration in eine komplette Bildverarbeitungsanwendung sind in von Hundelshausens Dissertation [8] nachzulesen.

Im Vornhinein sei noch kurz erwähnt, dass die Übertragbarkeit dieses Systems auf einen zukünftigen Einsatz in der humanoiden Kid-Size League nicht ohne Weiteres möglich ist, da es für den Einsatz mit einem omnidirektionalen Kamerasystem entwickelt wurde. Ein solches ist neuerdings in dieser Liga nicht mehr zulässig, was die in [9] festgestellte Effizienz des Verfahrens signifikant negativ beeinflussen könnte. Da es aber möglich ist, dieses zum Beispiel durch höhere Aufnahmeraten der Kamera oder langsamere Kamerabewegungen zu kompensieren, soll dieser Ansatz trotzdem im Folgenden kurz dargelegt werden.

Der vorgestellte Ansatz soll hauptsächlich der Detektion von Toren und Feldlinien dienen. Zur Erkennung der Feldlinien wird allerdings ein quasi inverses Vorgehen gewählt, welches nicht in erster Linie Feldlinien erkennt, sondern den Bodenbelag zwischen den Feldlinien. An den Rändern dieser Bereiche werden dann Feldlinien erwartet und identifiziert. Ein Effizienzgewinn soll dadurch erreicht werden, dass diese relativ großen Farbregionen verfolgt werden.

Das Verfolgen von Farbregionen läuft wie folgt ab. Aus dem vorhergehenden Bild stehen die Randpunkte der gesuchten Region zur Verfügung, wie sie dort detektiert

[7] URL: `http://www.fu-berlin.de/`

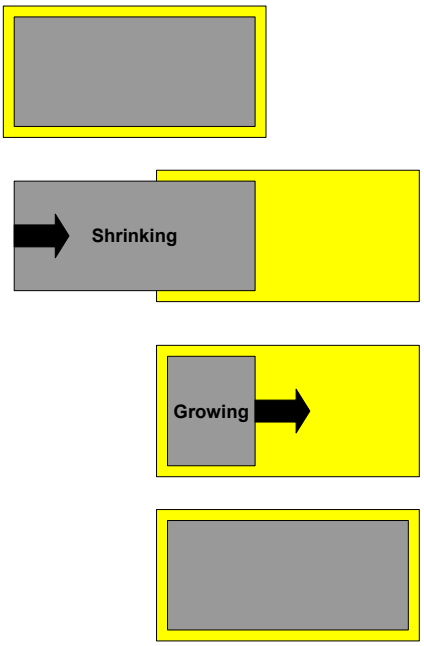


Abbildung 3.7: *Shrinking*- und *Growing*-Vorgang des Ansatzes der Freien Universität Berlin. Hier wird vereinfacht dargestellt, wie die Farbregion eines Tores verfolgt werden könnte, die sich leicht im Bild verschiebt.

wurden. Im aktuellen Bild werden sie als Ausgangspunkte benutzt. In einem ersten Schritt werden alle Randpunkte, die ein bestimmtes Kriterium nicht erfüllen, zum Innern der umschlossenen Region hin korrigiert. Dieses geschieht solange, bis sie das Kriterium erfüllen. Das Kriterium wird in der Regel sein, dass in ihrer Nachbarschaft ausreichend viele Pixel der gesuchten Farbe vorhanden sind. Das bedeutet, die Ausgangsregion wird auf den Schnittpunkt der Farbregion des vorherigen Bildes mit der Farbregion des aktuellen Bildes geschrumpft (*Shrinking*-Vorgang). Danach werden alle Randpunkte der aktuellen Region solange ausgedehnt, bis sie das Gütekriterium gerade noch erreichen (*Growing*-Vorgang). Damit ist die Farbregion im aktuellen Bild bestimmt worden (Abbildung 3.7 verdeutlicht den Vorgang noch einmal).

Dieses Vorgehen funktioniert allerdings nur (effizient), wenn sich die Farbregionen in auf einander folgenden Bildern (ausreichend) überschneiden beziehungsweise überhaupt in auf einander folgenden Bildern vorhanden sind. Ist dieses nicht der Fall, muss eine initiale Suche nach der entsprechenden Farbregionen gestartet werden, welche bei auffinden eines passenden Pixels nur noch den *Growing*-Vorgang startet. Der initiale Suchvorgang wird nicht genauer beschrieben (er hängt von der konkreten Anwendung ab). Hierbei kann aber berücksichtigt werden, dass bereits erkannte Farbregionen von der Suche ausgeschlossen werden können. Das verspricht wiederum einen Effizienzvorteil, vor allem wenn zuerst Regionen generiert werden, bei denen eine Überschneidung besteht.

Ergebnis dieses Ansatzes für den Einsatz in der Middle-Size League ist, dass pro Bild im Durchschnitt lediglich 10% der Pixel verarbeitet werden müssen. Probleme treten, wie schon erwähnt, bei der Portierung auf ein nicht-omnidirektionales Kamerasystem auf, da hier die Wahrscheinlichkeit von nicht oder wenig überlappenden Farbregionen steigt. Insbesondere gilt dieses für Objekte, die im Bild als wenige Pixel auftreten, entweder weil es kleinere Objekte (zum Beispiel der Ball) oder weiter entfernte sind.

3.4 Konzept der bisherigen Bildverarbeitung

3.4.1 Grundkonzept

Einen Ausgangspunkt für die aktuell verwendete Version der eingesetzten Bildverarbeitung liefert der Ansatz von Michael Stegbauer und Fredrik Mellgren [11], der im folgenden beschrieben wird.

Bei diesem werden aus dem gesamten Bild zunächst sogenannte *Runs* extrahiert. Ein *Run* beschreibt eine Zeile oder Spalte ähnlich dem RLE-Konzept (siehe hierzu auch Abschnitt 3.1.3). Allerdings werden hierbei jeweils ausschließlich die Position des ersten und des letzten Pixels derselben Farbe einer Zeile beziehungsweise einer Spalte gespeichert. Somit ergibt sich für jede betrachtete Zeile und Spalte eine Anzahl an *Runs*, welche maximal der Anzahl der unterschiedenen Farben ist. Die Bildfarbwerte der einzelnen Pixel werden anhand einer dreidimensionalen Farbtabelle den korrelierenden diskretisierten Farben zugeordnet.

Um den Rechenaufwand in Grenzen zu halten, werden lediglich die *Runs* für die i-te Zeile und j-te Spalte berechnet. Anstatt mit den *Runs* die jeweilige Farbe zu speichern, werden diese direkt dem zugehörigen Objekt zugewiesen (zum Beispiel Orange $\rightarrow$ Ball). Diese Struktur wird dann *RLE-Objekt* genannt.

In einem weiteren Schritt werden benachbarte *RLE-Objekte* zusammengefasst, die dem gleichen Objekt (zum Beispiel dem Ball) zugeordnet sind. Aus den nun entstandenen *RLE-Objekten* wird dann jeweils versucht, die Objekt-spezifischen Merkmale zu extrahieren. Für einen Ball sind diese beispielsweise Mittelpunkt und Radius.

3.4.2 Aktueller Ansatz

Im folgenden soll der aktuelle Ansatz zur Bildverarbeitung beschrieben werden, der bei den letzten Teilnahmen am *RoboCup* zum Einsatz kam. Die Beschreibung erfolgt hierarchisch vom aufrufenden zum aufgerufenen Modul. Im Allgemeinen findet die Ausführung so statt, dass der Erkennungsprozess das bildverar-

beitende Modul, den `GridImageProcessor`, in regelmäßigen Abständen ausführt. Dieser wiederum ruft die Objekt-spezifischen Erkenner wie zum Beispiel den `ClassifiedBallPerceptor` oder den `ClassifiedLinesPerceptor` auf, welche Objekte im Bild lokalisieren.

Bildvorverarbeitung

Als erstes initialisiert der `GridImageProcessor` alle angemeldeten Perzeptoren. Das bedeutet unter anderem, dass er bei jedem einzelnen die Container der sogenannten *Scan-Lines* registriert, so dass die Perzeptoren bei der späteren Objekterkennung auf diese zugreifen können. *Scan-Lines* sind spezielle Repräsentationen der einzelnen Zeilen und Spalten des Bildes, die bestimmte Informationen bereits extrahiert haben. Dieses vereinfacht die spätere Weiterverarbeitung und wird im Verlauf dieses Kapitels noch genauer ausgeführt werden.

Nach der Initialisierung werden bei jedem Aufruf des `GridImageProcessor`'s die folgenden Schritte ausgeführt. Das aktuelle Bild wird aus dem Puffer geladen und die internen Daten der Perzeptoren werden zurückgesetzt. Danach findet die Vorverarbeitung des Bildes mittels eines Rasters statt, bestehend aus vertikalen und horizontalen Linien konstanten Abstands (siehe Abbildung 3.1). So werden in diesem Schritt etwa zwei Fünftel des Bildes in *Scan-Lines* extrahiert. Dabei handelt es sich um vier verschiedene Arten von *Scan-Lines*, die unterschiedliche Informationen der Zeilen und Spalten repräsentieren:

1. Koordinaten der einzelnen Pixel
2. rohe Farbinformationen der einzelnen Pixel
3. Farbklassen der einzelnen Pixel
4. RLE-Kodierung der Farbklassen innerhalb einer Zeile oder Spalte

Das Generieren der *Scan-Lines* geschieht sukzessive. Als erstes werden die Koordinaten bestimmt. Mit Hilfe der Koordinaten-*Scan-Lines* werden *Scan-Lines* mit den rohen Farbinformationen erstellt, aus diesen dann farbklassifizierte *Scan-Lines* und zum Schluss die RLE-kodierten.

Sind die allgemeinen *Scan-Lines* erzeugt worden, bedarf es einer spezifischen Überarbeitung durch die einzelnen Perzeptoren. Sie filtern die *Scan-Lines* nach für den jeweiligen Erkenner interessanten Informationen und speichern diese direkt ab. Beispielsweise enthalten die *Scan-Lines* des Ball-Perzeptors nur *Runs* die als Ball-farben markiert sind. Dabei wird zusätzlich eine Rauschunterdrückung angewendet, so dass zum Beispiel einzelne Pixel, die von einer anderen Farbklasse umgeben sind, als diese Farbklasse gewertet werden. Während der Ball-Erkenner wieder RLE-*Scan-Lines* speichert, extrahiert der Linien-Perzeptor Punkte aus den *Scan-Lines*, an denen ein

Übergang von der Boden- zur Linienfarbe (oder umgekehrt) stattfindet. Das bedeutet, schon an dieser Stelle findet eine sehr unterschiedliche Weiterverarbeitung statt.

Wenn das Sammeln von Bildinformationen mittels Raster abgeschlossen ist, starten die Perzeptoren mit dem Erstellen der sogenannten Perzepte aufgrund der extrahierten Daten. Perzepte beschreiben erkannte Objekte anhand ihrer spezifischen Eigenschaften. Den Ball zum Beispiel beschreiben Radius und Position des Mittelpunktes bezüglich des Roboters eindeutig.

Im Folgenden sollen exemplarisch das Vorgehen des Ball- und des Linien-Erkenners beschrieben werden. Sie zeigen zum einen die Prinzipien, die angewendet werden, und zum anderen, inwiefern sich Perzeptoren unterscheiden müssen, um die unterschiedlichen Informationen aus einem Bild zu extrahieren.

Erkennen des Balls

Zur endgültigen Erkennung des Balls dienen die RLE-kodierten *Scan-Lines* als Grundlage, die in den vorherigen Schritten aus dem Bild generiert wurden und lediglich noch *Runs* der Ballfarbe enthalten.

Als erstes werden vertikale und horizontale *Runs* zu Komponenten zusammengefasst. Eine Komponente enthält folgende Informationen:

- die Liste aller beinhalteten *Runs*
- die Größe (diese entspricht der Summe der Länge aller beinhalteten *Runs*)
- die *Bounding-Box* (das minimale Rechteck mit parallelen Seiten zu den Bildachsen, das alle beinhalteten *Runs* einschließt)

Komponenten werden wie folgt gefunden. Als erstes wird aus einem noch nicht zugeordneten *Run* eine Komponente erzeugt. Ausgehend von diesem einen *Run* werden zunächst alle adjazenten *Runs* der Komponente zugeordnet. Ein *Run* ist adjazent zu einem anderen, wenn beide innerhalb der gleichen *Scan-Line* direkt auf einander folgen und einen geringen Abstand aufweisen oder wenn sie zu *Scan-Lines* gehören, die direkt neben einander liegen, und sich die *Runs* überschneiden. Abbildung 3.8 verdeutlicht, welche *Runs* in einem Schritt zusammengefasst werden. Dieses wird solange für die *Runs* der Komponente wiederholt, bis keine unzugeordneten *Runs* mehr erreicht werden. Zu beachten ist, dass in diesem Schritt nur *Runs* gleicher Orientierung zusammengefasst werden, also vertikale und horizontale getrennt.

Anschließend wird versucht, aus den drei größten vertikalen und den drei größten horizontalen Komponenten das Ball-Perzept zu erstellen. Sie werden jeweils paarweise betrachtet (eine vertikale und eine horizontale), angefangen mit den größten. Kann aus diesen kein Ball erkannt werden, gilt er als nicht vorhanden im aktuellen Bild.

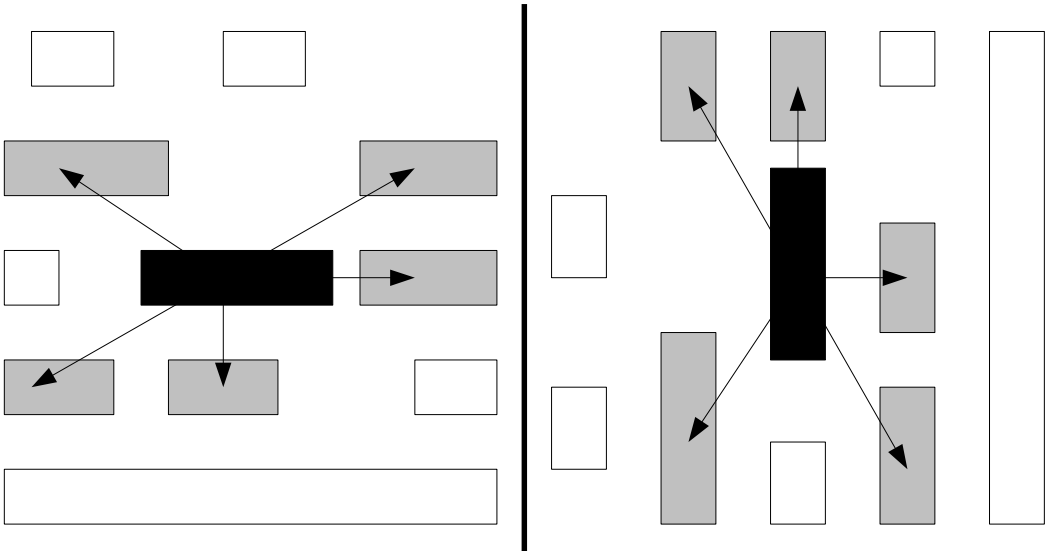


Abbildung 3.8: Beispiel für die Auswahl nächster *Runs* beim Erstellen von Komponenten ausgehend von einem *Run*. Alle dargestellten *Runs* sollen die gleiche Farbklasse besitzen. Die beiden schwarz markierten sind die Ausgangs-*Runs*. Graue *Runs* werden der Komponente zugeordnet und in die Bearbeitungsliste eingetragen. Weiße *Runs* werden in diesem Schritt nicht der Bearbeitungsliste hinzugefügt.

Zur Bestimmung des Perzepts werden als erstes alle Endpunkte von *Runs* der beiden betrachteten Komponenten klassifiziert im Bezug auf ihre Zugehörigkeit zum Rand des Balls. Unterschieden wird in:

- mit Sicherheit zu Ballrand gehörend
- eventuell zu Ballrand gehörend
- nicht zu Ballrand gehörend
- am Rand des Bildes liegend

Danach werden die vertikale und die horizontale Komponente zu einer verschmolzen, falls sie sich überschneiden. Ansonsten wird nur die größere von beiden im Folgenden betrachtet. Anhand der zum Ballrand gehörenden Pixel der resultierenden Komponente, wird versucht, den Umkreis des Balls zu errechnen. Dieses beinhaltet Radius und Position. Hierbei findet eine zusätzliche Farbklassifikation von Pixeln nahe der Ballrand-Pixel statt, um abzusichern, dass sie tatsächlich auf dem Rand des Balls liegen. Nur solche gehen in die Kalkulation des Ballumkreises ein.

Konnte der Umkreis des Balls im Bild bestimmt werden, und somit der Radius und die Position des realen Balls bezüglich des Roboters, werden diese Angaben einer Plausibilitätsprüfung unterzogen. Das nun extrahierte Ball-Perzept wird verworfen

(und die nächsten Komponenten betrachtet), falls eines der folgenden Kriterien zutrifft:

- Ballradius ist zu klein
- Ballradius ist zu groß
- Ballentfernung ist zu groß

In jedem anderen Fall, wird an dieser Stelle die Suche nach dem Ball erfolgreich beendet und das Ball-Perzept den an die Erkennung anschließenden Modulen zur Verfügung gestellt.

Erkennen von Feldlinien

Feldlinien werden grundlegend anders erkannt als der Ball. Das liegt schon daran, dass sie nicht als ein Haufen von Pixeln bestimmter Farbe auftreten, sondern eine kompliziertere Struktur aufweisen. Ein Bild kann zum Beispiel mehrere Feldlinien enthalten, die sich auch noch berühren können. Dieses muss erkannt und als unterschiedliche Linien registriert werden.

Im Wesentlichen besteht die Linienerkennung aus drei Schritten, dem Auffinden von Kanten, dem anschließenden Zusammensetzen dieser Kanten zu Feldlinien und dem extrahieren von Beziehungen zwischen den Kanten, wie zum Beispiel Schnittpunkte oder die (Teil-)Struktur eines Kreises.

Als erstes werden alle Pixel betrachtet, die an einem Übergang von Boden- zu Linienfarbe (oder umgekehrt) sitzen. Diese wurden bereits währen der Vorverarbeitungsphase erzeugt (siehe Abschnitt Bildvorverarbeitung). Sie werden miteinander verglichen und als ähnlich markiert, falls die Kanten an denen sie sitzen eine ausreichend ähnliche Steigung haben und der Abstand zwischen den beiden Pixeln klein genug ist. Anschließend wird für jeden potenziellen Kantenpunkt die Anzahl der ähnlichen Punkte bestimmt.

Danach werden ähnliche Punkte zu Kanten zusammengefasst, angefangen mit denen, die am meisten ähnliche Punkte besitzen. Punkte mit wenigen ähnlichen werden verworfen. Das heißt, nach diesem Schritt kann auf Grundlage von Kanten gearbeitet werden und nicht mehr auf einzelnen Pixeln. Analog des Zusammenschlusses von einzelnen Punkten zu Kanten, werden die Kanten miteinander verglichen und wenn möglich zu längeren zusammengefasst.

Im Folgenden werden die Kanten zu Feldlinien kombiniert. Dazu werden Kantenpaare betrachtet und deren Orientierung und Abstand zu einander verglichen. Aus den Kantenpaaren, die am besten zu einander passen und ein Mindestmaß an Korrelation aufweisen, werden Linien erstellt (siehe Abbildung 3.4.2). Diese stehen dann den anschließenden Modulen (zum Beispiel der Selbstlokalisierung) zur Verfügung. Zum

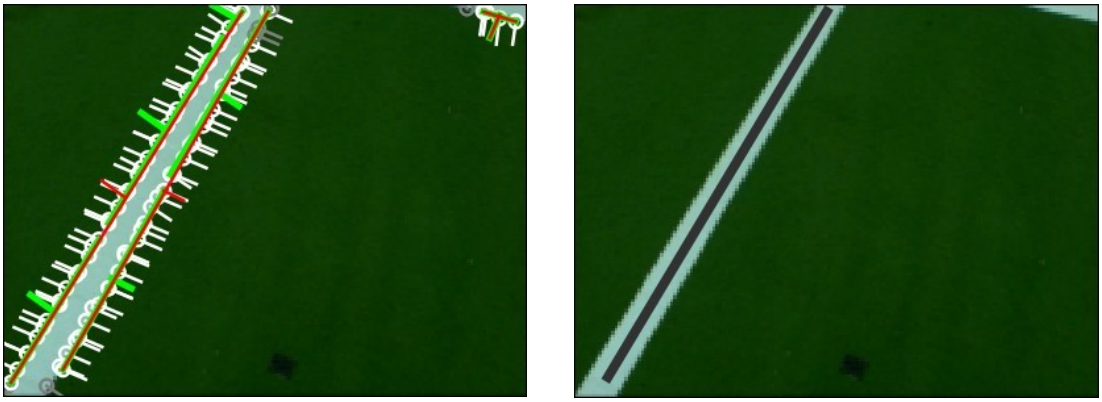


Abbildung 3.9: Darstellung zur Rekonstruktion einer Feldlinie nach dem bisherigen Ansatz der Darmstadt Dribblers. Links sind die Pixel dargestellt, die zu den Kanten gehörig erkannt wurden. Zusätzlich wird jeweils ein zur Kantensteigung orthogonaler Vektor angezeigt. Rechts ist das resultierende Perzept eingezeichnet.

Schluss werden aus den erkannten Feldlinien Kreuzungen und der Mittelkreis extrahiert und ebenfalls als Perzepte bereitgestellt.

3.4.3 Vor- und Nachteile

Ein Vorteil des aktuellen Bildverarbeitungsansatzes ist, dass nach der Vorverarbeitung, das heißt nach dem bilden längenkodierter Farbinformationen, das gesamte Bild in kompakter Form beschrieben wird. Das ermöglicht eine genaue Erkennung der Objekte in den folgenden Schritten.

Allerdings geht dieser Informationsgewinn auf Kosten der Performanz, da jedes neue Bild komplett in RLE-Objekte segmentiert werden muss. Hier setzt der Ansatz der ActiveVision an, der versucht nur noch Teile des Bildes zu klassifizieren.

4 Anforderungen und Ziele

In diesem Kapitel werden die Anforderungen und Ziele präsentiert, die es gilt mit dem Ansatz der ActiveVision zu erfüllen. Diese gehen von der Verwendbarkeit über die Zuverlässigkeit bis hin zur Effizienz des Systems.

4.1 Integration in die bisherige Anwendung

Ein sehr wichtiger Punkt bei der Umsetzung des ActiveVision-Ansatzes ist die Integration in die bestehende Anwendung zur Steuerung eines Roboters im *RoboCup*. Hierbei müssen die Vorgaben des zugrunde liegenden Frameworks und die Anforderungen der *RoboCup*-Anwendung an die Bildverarbeitung berücksichtigt werden.

4.1.1 *RoboFrame*

Die Anwendung zur Steuerung des Roboters, die vom Team der Darmstadt Dribblers entwickelt und eingesetzt wird, basiert auf dem von Sebastian Petters und Dirk Thomas entworfenen Framework. In Einzelheiten wird es in „RoboFrame – Softwareframework für mobile autonome Robotersysteme" [13] beschrieben. Kurz zusammengefasst bietet es ein Grundgerüst, um möglichst Plattform unabhängige und flexible Anwendungen für mobile Roboter zu entwerfen. Hierbei wird erwartet, dass sich die konkrete Anwendung aus mehreren nur leicht gekapselten Modulen zusammensetzt. Eine komfortable und effiziente Kommunikationsmöglichkeit zwischen den Modulen wird vom Framework bereitgestellt. Ein Modul der gesamten *RoboCup*-Anwendung ist unter anderem auch die Bildverarbeitung.

Damit sich ein Modul, und damit auch die ActiveVision, problemlos in die übergeordnete Applikation einfügt, muss es mit anderen Modulen über die vorgegeben Wege kommunizieren. Innerhalb eines Prozesses geschieht dieses über Puffer, zwischen Prozessen über *Blackboards*. Das Modul definiert gegenüber dem Framework, welche Puffer beziehungsweise *Blackboards* es für eingehende Daten verwendet und über welche es seine berechneten Daten ausgibt. Von welchen konkreten Modulen die Eingangsdaten kommen oder die Ausgangsdaten verwendet werden, ist hierbei unerheblich und dem einzelnen Modul nicht bekannt. Dieses gewährleistet die Flexibilität,

dass Module zum einen einfach durch andere mit gleichem Kommunikationsinterface ersetzt und zum anderen leicht über mehrere Prozesse verteilt werden können.

Des Weiteren müssen Module neben den Konstruktoren eine zusätzliche Initialisierungsmethode bereitstellen. Diese ist notwendig, um die generischen Kommunikationswege nach der Initialisierung eines Moduls einrichten und danach das Erstellen des Moduls abschließen zu können. Zur Ausführung muss eine Methode `execute` zur Verfügung gestellt werden, mit der das Modul in mehr oder weniger regelmäßigen Abständen ausgeführt werden kann.

Um die Ressourcen der Gesamtanwendung nicht unnötig zu strapazieren, sollte ein Modul außerdem vor dem Ausführen bestimmter Berechnung prüfen, ob die entstehenden Daten überhaupt angefragt werden. Auch hierfür stellt das *RoboFrame* einen Mechanismus zur Verfügung, den es zu benutzen gilt. Zum Beispiel sollte man den Aufruf spezifischer Erkenner davon abhängig machen, ob die generierten Objekt-Perzepte von einem anderen Modul angefordert werden.

4.1.2 *RoboCup*-Anwendung

Aus der Applikation[1] zur Steuerung des Roboters ergeben sich weitere Anforderungen an die Implementierung der ActiveVision. Sie soll sich, wie schon im vorherigen Abschnitt für Module im Allgemeinen angesprochen, problemlos in die Komplettanwendung integrieren, indem sie einfach das bisherige Bildverarbeitungsmodul ersetzt. Hierfür gilt es gewisse Richtlinien einzuhalten, damit andere Module nicht angepasst werden müssen.

Zum einen muss die ActiveVision die gleichen Ausgabewege für die bereitgestellten Daten verwenden, wie dieses schon bei der aktuellen Bildverarbeitung geschieht, und zum anderen müssen die Ausgabedaten selbst in gleicher Form repräsentiert werden. Das bedeutet zum Beispiel, dass das Perzept für den erkannten Ball weiterhin seinen Mittelpunkt in Koordinaten in Bezug auf das Bild und das Roboterkoordinatensystem und den Radius im Bild und in der Realität angibt. Dadurch wird gewährleistet, dass für alle weiterverarbeitenden Module die benötigten Informationen zur Verfügung stehen. Natürlich wäre eine Erweiterung der Perzepte, die die Objekte repräsentieren, möglich, solange sie die vorherigen Daten in gleicher Form bereitstellen. Darauf wurde aber verzichtet, da sich hierfür keine Notwendigkeit gezeigt hat. Es werden also die „alten" Perzepte weiter verwendet.

Eine zusätzliche Anforderung aus der *RoboCup*-Anwendung ist, dass die Bildverarbeitung auf der durch die Farbtabelle bereitgestellten Klassifikation von rohen Farbwerten arbeitet. Deshalb verwendet der ActiveVision-Ansatz ausschließlich diese Informationen. Das heißt, hier werden keine rohen Farbwerte mehr berücksichtigt (im

[1] Einen Überblick über die *RoboCup*-Anwendung der Darmstadt Dribblers bietet [6]

Gegensatz zum bisherigen Konzept). Das macht ihn besonders flexibel, da Farbänderungen relevanter Objekte oder im Spielfeld (zum Beispiel aufgrund von Regeländerungen) dadurch eingearbeitet werden können, dass die neuen Farben anhand der Farbtabelle passend klassifiziert werden.

4.2 Hohe Anpassungsfähigkeit und Flexibilität

Schon im vorherigen Abschnitt wurde ein Designkriterium erwähnt, welches eine hohe Flexibilität bereitstellen soll. Zusätzlich zur Unabhängigkeit von rohen Farbwerten, ist ein Ziel der ActiveVision Optimierungsmöglichkeiten und Anpassungsfähigkeit auch nach ihrer vorläufigen Fertigstellung bereitzuhalten. Da gerade Bildverarbeitungssysteme von Expertenwissen profitieren können, welches immer weiter vorangetrieben wird und in der Regel zu neuen Erkenntnissen kommt, soll die Umsetzung der ActiveVision möglichst parametrisierbar gestaltet werden. Das bedeutet, jegliche sinnvollen Variablen sollen auch noch im Nachhinein leicht adaptiert werden können, um die Anwendung weiter zu optimieren oder eventuell an neue Gegebenheiten anzupassen.

Als eine sinnvolle und womöglich benötigte Einstellung sei hier beispielhaft genannt, den (minimalen/maximalen) Abstand von Suchpunkten zu verändern. Dieses wird wahrscheinlich von Nöten sein, wenn eine Kamera mit einer veränderten Auflösung eingesetzt werden soll.

4.3 Grad der Genauigkeit

Damit die an die Bildverarbeitung angeschlossenen Module möglichst effektiv arbeiten können, muss schon durch die Bildverarbeitung ein hohes Maß an Genauigkeit und Zuverlässigkeit der Ausgabedaten gewährleistet werden. Sollte diese Anforderung verletzt werden, multiplizieren sich die anfänglichen Ungenauigkeiten eventuell zu erheblichen Fehlfunktionen auf. Wird zum Beispiel der Mittelkreis des Feldes an einer ganz anderen Stelle angenommen, als er sich tatsächlich befindet, kann das die Selbstlokalisierung des Roboters stark negativ beeinträchtigen. Solche und ähnliche Fehler könnten im schlimmsten Fall zur Folge haben, dass auf das eigene anstatt auf das gegnerische Tor geschossen wird.

4.4 Behandlung von Bildfehlern (Rauschen)

In den aufgenommenen Bildern kommt es regelmäßig zu Fehlern, insbesondere nach der Klassifikation einzelner Pixel. Das bedeutet, dass beispielsweise Pixel, die eigentlich zum orangefarbenen Ball gehören, als gelb klassifiziert werden. Diese Bildunreinheiten müssen von einer Bildverarbeitung berücksichtigt und angemessen behandelt werden. Wäre dieses nicht der Fall, könnte jeder Fehler im Bild fatale Auswirkungen haben. Zum Beispiel könnte ein Ball im gelben Tor erkannt werden, weil eigentlich gelbe Pixel des Öfteren Orange zugeordnet werden. Bildfehler sollten also weder dazu führen, dass Objekte falsch oder garnicht erkannt werden, obwohl sie ausreichend im Bild vorhanden sind, noch dazu, dass nicht im Bild vorhandene Objekte als präsent gewertet werden.

4.5 Erkennung von verdeckten Objekten

Auch die Verdeckung von Objekten ist ein häufiges Problem im *RoboCup*. In der Regel wird es vorkommen, dass ein oder mehrere Roboter das Tor des Gegners aber auch das eigene nur teilweise erblicken lassen. Außerdem kann durch die Konstruktion des Roboters nicht ausgeschlossen werden, dass eigene Körperteile einen nahe gelegenen Ball partiell verdecken. Besonders in diesem Fall wäre es sehr fahrlässig, den Ball nicht erkennen zu können, da es den Spieler eines enormen Vorteils beraubt. Schlussfolgerung ist, dass die ActiveVision auch Objekte erkennen können sollte, die nur teilweise zu sehen sind, was übrigens auch bei Objekten am Bildrand auftritt.

4.6 Erkennung von nahen und fernen Objekten

Schon die aktuelle Bildverarbeitung kann natürlich nahe und weiter entfernte Objekte erkennen. Den Ansatz der ActiveVision könnte man aber dazu nutzen, Objekte in noch größerer Entfernung zu erkennen, ohne dabei die Gesamtperformanz signifikant beeinträchtigen zu müssen. Gerade wenn die Auflösung der Bilder erhöht wird, ergeben sich daraus neue Möglichkeiten, die es auszuschöpfen gilt.

Im konventionellen Ansatz müsste man den konstanten Abstand der Suchrasterlinien verringern, um weiter entfernte Objekte mit großer Wahrscheinlichkeit nicht zu überspringen. Das bedeutet allerdings, dass auch bei der Bildanalyse von Aufnahmen nahe des Roboters ein unnötig enges Gitter zugrunde gelegt wird. Das vermindert die Effizienz des Ansatzes signifikant.

Die ActiveVision hingegen kann ausnutzen, dass ihre Suchstrategie nicht auf einem starren Raster basiert. Das heißt, hier sollte ausgenutzt werden, dass Suchpunkte nahe

des Roboters einen relativ großen Abstand haben können, wohingegen Punkte, die projiziert weit vom Roboter entfernt liegen, so nahe bei einander lokalisiert sind, wie es die Auflösung zulässt und es für sinnvoll erachtet wird.

4.7 Effizienzgewinn gegenüber der aktuellen Bildverarbeitung

Eines der Hauptziele des ActiveVision-Ansatzes ist es, die Effizienz der aktuell bei den Darmstadt Dribblers eingesetzten Bildverarbeitung (siehe Abschnitt 3.4) signifikant zu steigern. Die Grundidee liegt hierbei darin, dass es möglich sein sollte, wesentlich weniger Pixel des Bildes analysieren und dennoch ähnlich aussagekräftige Informationen aus dem Bild extrahieren zu können. Das heißt, die Erkennungsgenauigkeit soll möglichst beibehalten oder sogar gesteigert werden, und trotzdem eine bemerkbare positive Einflussnahme auf die Ausführungszeiten der Bildverarbeitung stattfinden.

5 Konzept

In diesem Kapitel wird das Konzept der ActiveVision beschrieben, welches sich an den in Kapitel 4 beschriebenen Anforderungen und Zielen orientiert. Besonderer Schwerpunkt dabei ist es, die Voraussetzungen für eine Ausführungsbeschleunigung der Bildverarbeitung zu erfüllen. Des Weiteren soll sich die Anwendung reibungslos in die bestehende *RoboCup*-Applikation einfügen und keinen Nachteil bezüglich der Erkennungszuverlässigkeit ergeben.

Im Folgenden werden daher kurz das Bildverarbeitungsmodul selbst und die von diesem ausgeführten Perzeptoren beschrieben, welche für die Erkennung verschiedener Objekte oder der Feldlinien zuständig sind. Vorab soll allerdings eine Abgrenzung zu anderen Konzepten stattfinden, welche sich ebenfalls unter dem Begriff „Active Vision" beziehungsweise „Aktives Sehen" wiederfinden.

5.1 Abgrenzung des „Active Vision"-Begriffs

Der Begriff „Active Vision" wird in der aktuellen Forschung für verwandte aber doch unterschiedliche Konzepte im Rahmen der Signalverarbeitung benutzt. Dieser Abschnitt soll verdeutlichen, welche anderen Ansätze noch unter der Bezeichnung des „Aktiven Sehens" oder „Aktiven Wahrnehmens" zu finden sind, und wie sich der in dieser Arbeit verwendete Begriff der „Active Vision" davon unterscheidet.

5.1.1 Aktive Sensoren

Die ersten Gruppe von Konzepten, die sich unter der etwas allgemeineren Bezeichnung der „Aktiven Wahrnehmung" wiederfindet, bezieht sich darauf, dass die eingesetzten Sensoren aktive Sensoren sind. Das bedeutet, sie fangen nicht einfach natürliche Daten der Umwelt ein, sondern senden Daten aus und messen deren Verhalten. So gelten zum Beispiel folgende Sensoren als aktiv:

- Laserscanner
- Ultraschallsensoren
- Tastsensoren

Sie sind mit dem Ansatz der ActiveVision nur insofern verwandt, dass teilweise Bildähnliche Signale verarbeitet und daraus Objekte erkannt werden müssen.

5.1.2 Aktive Kamerasteuerung

Einen kurzen Überblick über die Inhalte der zweiten Gruppe von Ansätzen und über weiterführende Literatur gibt [12]. Hier wird explizit der Begriff des „Aktiven Sehens" benutzt. Dieser wird dadurch begründet, dass die eingesetzten Kameras zur Aufnahme von Bildern aktiv gesteuert werden. Das heißt, sie erfassen nicht mehr oder weniger willkürliche Daten ihrer Umwelt, sondern fokussieren gezielt bestimmte Positionen. Dieses geschieht aufgrund bestimmter Anforderungen an das aufzunehmende Bild.

So kann ein Robotersystem nach der an dieser Stelle verwendeten Definition des „Aktiven Sehens" zum Beispiels wie folgt vorgehen, um einen Reifen an ein Auto zu montieren. In einem ersten Schritt wird das gesamte Rad erfasst und die Schrauben zur Befestigung identifiziert. Um ein entsprechendes Werkzeug aber exakt platzieren zu können, fokussiert das verwendete Kamerasystem die Schraube, die als nächstes angezogen oder gelöst werden soll. Den konkreten Ansatz eines „aktiven" Bildverarbeitungssystems und auch das aufgezeigte Beispiel kann der Arbeit von Ulrich Büker [3] entnommen werden.

5.1.3 „Aktives Sehen" im Rahmen der ActiveVision

Beim in dieser Arbeit vorgestellten Konzept der ActiveVision rechtfertigt sich die Bezeichnung als „aktives" Bildverarbeitungssystem dadurch, dass die durch eine Kamera aufgenommenen Bilder aktiv verarbeitet werden. So werden beispielsweise bei der Erkennung von Objekten wie dem Ball Daten aus der Modellierung berücksichtigt. Diese Informationen dienen der Initialisierung eines Anfangspunktes bei der Suche nach dem Objekt, das heißt, die Suche beginnt an der Stelle im Bild, an der die aktuelle Position des Objektes vermutet wird.

Außerdem können die Abstände von im Bild analysierten Pixeln an die Größe des gesuchten Objektes angepasst werden. Je nachdem welche Ausmaße das zu erkennende Objekt im aktuellen Bild annehmen kann, werden größere oder kleinere Abstände gewählt. Diese können auch innerhalb des Bildes noch variieren, um zu berücksichtigen, dass unterschiedliche Regionen im Bild in der Realität unterschiedlich weit vom Roboter entfernt sind.

Konnte ein Punkt im Bild gefunden werden, der vermutlich zum gesuchten Objekt gehört (in der Regel ist dieses über die Farbe des Pixels zu bestimmen), wird von diesem ausgehend versucht, das Objekt zu identifizieren. Ist dieses gelungen, kann die Suche an dieser Stelle abgebrochen werden und die restlichen Teile des Bildes müssen

durch den aktuellen Perzeptor nicht verarbeitet werden.

Somit hängt die Effizienz der Objekterkennung stark damit zusammen, wie exakt die Daten aus der Modellierung mit dem aktuellen Zustand der realen Umwelt übereinstimmen. Das hängt zum einen von der Modellierung selbst ab, zum anderen aber auch von dem Verhalten der Umwelt. So kann ein Gegenstand, der gerade erst seinen Zustand von „unbewegt" in „bewegt" geändert hat (beispielsweise ein geschossener Ball), zumindest in den ersten Momenten natürlich schlecht modelliert werden. Welche Auswirkungen diese Erkenntnis auf den Einsatz der ActiveVision in der Praxis hat, wird Kapitel 7 aufzeigen.

Ein etwas anderer Ansatz innerhalb der ActiveVision wird zur Erkennung von Feldlinien herangezogen. Hier werden Anfangssuchpunkte nicht über die Modellierung definiert, sondern selbstständig gefunden, indem ein gewisses Expertenwissen implementiert wird. Dieses Expertenwissen besagt, dass Feldlinien, die innerhalb eines Bildes zu sehen sind, immer auch den Rand des Bildes schneiden. Das ist dem Öffnungswinkel der Kameras und dem eingeschränkten Bewegungsradius des Roboters geschuldet. Dadurch können Ausgangspunkte für die Konstruktion von Feldlinien gefunden werden, indem der Bildrand nach eben den Schnittpunkten mit Feldlinien untersucht wird. Ausgehend von diesen werden dann die Feldlinien innerhalb des Bildes identifiziert.

5.2 Bildverarbeitungsmodul

Das `ActiveImageProcessor` genannte Bildverarbeitungsmodul wird von der übergeordneten Anwendung in regelmäßigen Abständen aufgerufen. Aufgabe des `ActiveImageProcessor`'s ist es, hauptsächlich die einzelnen Erkenner (auch Perzeptoren genannt) zu initialisieren, zu koordinieren und auszuführen. Dazu werden wenige Schritte zur Bildvorverarbeitung angestellt, wie beispielsweise das Ermitteln einer Horizont-ähnlichen Begrenzung im Bild. Diese drückt aus, ab welcher Zeile des Bildes die dargestellten Regionen in der Regel nicht weiter verarbeitet werden müssen. Eine einfache Regel, um diese Begrenzung zu definieren, ist, dass Bereiche, die eine gewisse Entfernung zum Roboter überschreiten uninteressant sind (weil sie zum Beispiel nicht mehr innerhalb des Spielfeldes liegen können).

Indem das Bildverarbeitungsmodul die einzelnen Erkenner für die unterschiedlichen relevanten Objekte steuert, erfüllt es für die Gesamtapplikation die Funktion, aus gegebenen Bildern aussagekräftige Daten zu extrahieren. De facto geschieht das allerdings nicht innerhalb des Moduls selbst, sondern durch den Aufruf der Perzeptoren, deren Konzept im folgenden Abschnitt beschrieben wird. Die bereitgestellten Informationen werden entsprechend der Kommunikationswege des zugrunde liegenden Frameworks und den Anforderungen zur reibungslosen Integration bereitgestellt (siehe Abschnitt

4.1)

Derzeit ist die Reihenfolge irrelevant, in der die einzelnen Perzeptoren ausgeführt werden. Wie aber in Abschnitt 8.2 gezeigt werden wird, könnte das für die mögliche Weiterentwicklung der ActiveVision eine Rolle spielen, falls Perzeptoren untereinander Informationen austauschen. Dann könnte es Sinn machen, bestimmte Perzeptoren vor anderen ablaufen zu lassen. Eine bestimmte Ausführungsordnung ist also momentan nicht berücksichtigt, aber es besteht die Möglichkeit, nicht jeden registrierten Perzeptor während eines Durchlaufs zu starten. Es werden lediglich diejenigen ausgeführt, deren Perzepte von anderen Modulen angefordert werden. Dieses ermöglicht es, schon in übergeordneten Modulen dem Konzept der ActiveVision Rechnung zu tragen, indem nur tatsächlich aktuell relevante Informationen geordert werden.

Während die Perzeptoren die einzelnen Objekte identifizieren, dient diesen der `ActiveImageProcessor` seinerseits wiederum als Kommunikationsschnittstelle zu den Informationen, die das konkrete Bild betreffen. Er stellt als Spezialisierung des `ImageHandler`'s Methoden zur Verfügung, mit denen zum Beispiel Farbklassen zu Pixeln innerhalb des Bildes ermittelt werden können. Somit benötigen die Perzeptoren keinen Zugriff auf das Bild. Dieser wird zentral über das Bildverarbeitungsmodul geregelt.

5.3 Perzeptoren

Perzeptoren dienen dem Erkennen einzelner relevanter Objekte. Wie beschrieben, werden sie vom `ActiveImageProcessor` sequenziell gestartet. Beim Ansatz der ActiveVision kann zwischen zwei verschiedenen Konzepten unterschieden werden. Das eine dient dem Erkennen von Objekten, und durch das andere wird die Identifikation der Feldlinien und der daraus bestehenden Kreuzungen und des Mittelkreises umgesetzt.

Als erstes sollen die Gemeinsamkeiten beider Konzepte beschrieben werden. Da in der humanoiden Kid-Size League im *RoboCup* sämtliche relevanten Objekte mit strikt definierten Farben versehen sein müssen, bilden die Farbinformationen eines Bildes die wichtigste Grundlage der Bildverarbeitung. Alle Perzeptoren arbeiten ausschließlich auf Farbklassen. Die rohen Farbwerte des Bildes, die durch eine Farbtabelle den Farbklassen zugeordnet werden (siehe Abschnitt 3.1.1), sind für sie nicht sichtbar. Dieses macht die Erkenner robust gegenüber Veränderungen, die das Farbvorkommen in der realen Umgebung betreffen (zum Beispiel durch Lichtverhältnisse oder Regeländerungen im *RoboCup*). Solche Modifikationen können dann relativ einfach durch ein Anpassen der Farbtabelle umgesetzt werden. Weitere Flexibilität gewinnen die Perzeptoren, aber teilweise auch der `ActiveImageProcessor`, durch eine tiefgreifende Parametrisierung. So sind alle relevanten und sinnvollen Parameter leicht zur Laufzeit

einstellbar. Zum Beispiel kann der minimale Abstand, den zwei Suchpunkte innerhalb des Bildes nicht unterschreiten dürfen variiert werden. Dieses dient der Anpassung und Optimierung der Gesamtanwendung an sich ändernde Umstände (bezüglich Spielfeld und Roboter) und eventuell neue Erkenntnisse.

Die Erkennungszuverlässigkeit – das heißt, genaues Erkennen von (verdeckten) Objekten in Bildern (mit Rauschen) – der bisher Implementierten Erkenner zur Ball- und Feldliniendetektion ist hoch (für genauere Ergebnisse siehe Abschnitt 7.1). Das ergibt sich jedoch nicht zwangsweise durch das Konzept der ActiveVision, welches sich in erster Linie auf einen Effizienzgewinn konzentriert, sondern aus der Rekonstruktion der Objekte oder Feldlinien aus den extrahierten Daten, bei der Expertenwissen sinnvoll angewendet wurde. Allerdings lässt sich ebenfalls ein entstandener Vorteil bei den Ausführungszeiten wiederum in genauere und zuverlässigere Erkennungsmuster investieren. So ist es zum Beispiel möglich, in unteren Regionen des Bildes größere Abstände zwischen den Suchpunkten zu wählen, während man im oberen Teil so kleine Schritte wählt, dass auch noch minimale Objekte gefunden werden können.

Im Folgenden wird nun das Konzept der Objekt-Erkenner vorgestellt. Hierbei initialisiert der Erkenner einen Ausgangspunkt der Suche anhand der Daten aus der Modellierung. Diese bieten eine Schätzung der aktuellen Position des gesuchten Objektes, welche in das aktuelle Bild projiziert werden kann. Sich von dieser Anfangsposition ausbreitend wird das Bild nach dem Objekt durchsucht. Sobald es gefunden wurde (das heißt, der Fund gilt auch als zuverlässig), werden die entsprechenden Informationen den anschließenden Modulen über Puffer zur Verfügung gestellt und die Suche nach diesem Objekt endet an dieser Stelle. Das kann dazuführen, das nur ein relativ geringer Teil des Bildes vom aktuellen Perzeptor durchsucht werden muss. Ob die im Bild identifizierte Region tatsächlich dem gesuchten Objekt zuzuordnen ist, kann unter anderem dadurch überprüft werden, dass die erkannten Ausmaße mit den realen verglichen werden. Auch dieses gehört explizit zum Konzept der ActiveVision, welches die Verwendung von solchen Metainformationen vorsieht.

Abschließend wird noch die Vorgehensweise des Feldlinien-Erkenners beschrieben. Sie unterscheidet sich aus dem Grund, dass Feldlinien eine prinzipiell andere Struktur aufweisen als einzelne Gegenstände. Sie müssen als einzelne gerade Strecken, Kreuzungen und Mittelkreis erkannt werden und können nicht als ein großes Ganzes gesehen werden, wie beispielsweise ein Tor, das einfach durch seine oberen, unteren, linken und rechten Enden definiert werden kann. Das erschwert das Festsetzen von einzelnen wirklich sinnvollen Ausgangssuchpunkten. Außerdem haben Feldlinien in ihrer Breite die geringsten Ausmaße aller relevanten Objekte, was leichter dazu führt, dass an einer vorhergesagten Position keine Feldlinie zu finden ist. Durch eine gesonderte Strategie kann also das schon angesprochenen Expertenwissen ausgenutzt werden, bei dem Linienschnittpunkte mit den Bildrändern gesucht werden (siehe Abschnitt 5.1.3). Ausgehend von diesen Startpositionen, wird dann das komplette Netz von Feldlinien innerhalb des Bildes rekonstruiert. Auch hierbei werden wieder Daten über die realen

Ausmaße von Feldlinien berücksichtigt.

Eine Umsetzung des Linien-Perzeptors auf Basis des Ansatzes der Objekt-Erkenner wäre allerdings auch möglich und kann bei der Weiterentwicklung der ActiveVision in Betracht gezogen werden, wenn es sich als sinnvoll erweist (zum Beispiel, weil das Vorkommen von Feldlinien eingeschränkt wird).

6 Realisierung

Die Implementierung des ActiveVision-Ansatzes ist komplett in C++ realisiert worden und integriert sich reibungslos in die Gesamtanwendung zur Steuerung eines humanoiden Roboters im *RoboCup*. Zu diesem Zweck wurde innerhalb der Applikationslösung ein neues Projekt namens `ActiveVisionApp` angelegt, welches soweit möglich und sinnvoll alle Modifikationen enthält, die im Rahmen der ActiveVision durchgeführt wurden. Ausnahmen stellen eine Anpassung der Kamera-Klassen dar und natürlich die Einbindung in die *RoboCup*-Software, die an den geeigneten Stellen stattzufinden hat. So müssen zum Beispiel neu definierte Schlüssel registriert und das neue Bildverarbeitungsmodul bei der Applikationsinitialisierung einem Prozess zugeordnet werden[1]. Außerdem wurde ein neuer Namensraum (*namespace*) eingeführt, in dem die ActiveVision-Komponenten definiert sind: `activevision`.

Die Beschreibung der Umsetzung unterteilt sich wie folgt. Als erstes werden die schon erwähnten Änderungen an den Kamera-Klassen beschrieben und im Anschluss die speziell für die ActiveVision eingeführten Klassen, die bestimmte geometrische Konstrukte und Funktionen zur Verfügung stellen. Danach wird die Implementierung des Bildverarbeitungsmoduls dargestellt und abschließend die Realisierung der Perzeptoren.

6.1 Erweiterung der Kamera-Klassen

Zum Zweck einer effizienten Projektion von Positionen bezüglich des Roboterkoordinatensystems in Bildkoordinaten wurden die Kamera-Klassen der *RoboCup*-Applikation angepasst. Dazu wurde im Interface `ICamera`, das die Basis aller Kamera-Klassen darstellt, die Methode `getImagePosition` eingeführt, welche von den konkreten Kamera-Realisierungen implementiert werden muss. Einziger Parameter der Funktion ist der dreidimensionale Sichtstrahl vom Kameramittelpunkt zur realen Position (angegeben in Bezug auf das Kamerakoordinatensystem). Dieser genügt, um die entsprechenden Bildkoordinaten zu bestimmen, indem er mit der Bildebene geschnitten wird.

Ein solches Vorgehen ist sinnvoll, da es zum einen innerhalb der Kamera-Klassen

[1] Der hier gewählte Prozess entspricht dem, dem auch die bisherige Bildverarbeitung zugeteilt war.

direkt Typ-spezifische Besonderheiten, wie beispielsweise Verzerrung, berücksichtigen und zum anderen ohne großen Mehraufwand umgesetzt werden kann. Eine konkrete Implementierung wird im folgenden beschrieben. Sie ist in der Klasse `CameraWithDistortion` implementiert und liefert somit die gewünschte Funktion für Kameras mit Verzerrung (engl.: distortion), wie sie derzeit im Einsatz sind.

Die konkrete Umsetzung ist relativ unkompliziert. Folgende Kameratyp-spezifischen Werte werden dazu benötigt. Zum einen der optische Mittelpunkt (C_{img}), gegeben in Bildkoordinaten, und zum anderen der horizontale (α_{img}) und vertikale (β_{img}) Öffnungswinkel der Kamera. Aus dem übergebenen Sichtstrahl werden ebenfalls dessen horizontaler (α_{ray}) und vertikaler (β_{ray}) Winkel extrahiert. Die Winkel beziehen sich auf eine zur Bildebene normalen Achse. Die Postion der realen Koordinaten im Bild (P_{img}) kann dann nach (6.1) berechnet werden. Bei dieser Berechnung werden einfach die Winkel des Sichtstrahls mit den Öffnungswinkeln in Relation gesetzt, und entsprechend die gesuchte Position im Bild in Bezug zum Bildmittelpunkt bestimmt. Abbildung 6.1 soll diesen Zusammenhang etwas verdeutlichen.

$$P_{img} = C_{img} + \begin{pmatrix} \frac{\alpha_{ray}}{\alpha_{img}} & 0 \\ 0 & \frac{\beta_{ray}}{\beta_{img}} \end{pmatrix} \cdot C_{img} \tag{6.1}$$

Diese Funktionalität ist eine grundlegende Voraussetzung um den Ansatz der ActiveVision umsetzen zu können, welcher für die Objekterkennung reale Koordinaten in Bild-bezogene übersetzen muss, um einen Ausgangssuchpunkt initialisieren zu können.

6.2 Einführung spezieller Geometrie-Klassen und -Methoden

In diesem Abschnitt sollen eingeführte Geometrie-Klassen und deren Funktion beschreiben werden. Sie sind alle im Verzeichnis `geometry` des ActiveVision-Projektes zu finden. Als erstes werden die Klassen dargestellt, die tatsächlich geometrische Gebilde verkörpern, wie einen Kreis (*Circle*), eine begrenzte gerade Strecke (*Line*), eine Kreuzung von Strecken (*Crossing*) oder eine begrenzte Kurve, definiert durch eine geordnete Menge von Punkten (*Curve*). Anschließend wird das Prinzip eines *Runs* erklärt. Zum Schluss folgt die Darstellung allgemeiner geometrischer Methoden, welche von ActiveVision-Komponenten benötigt werden.

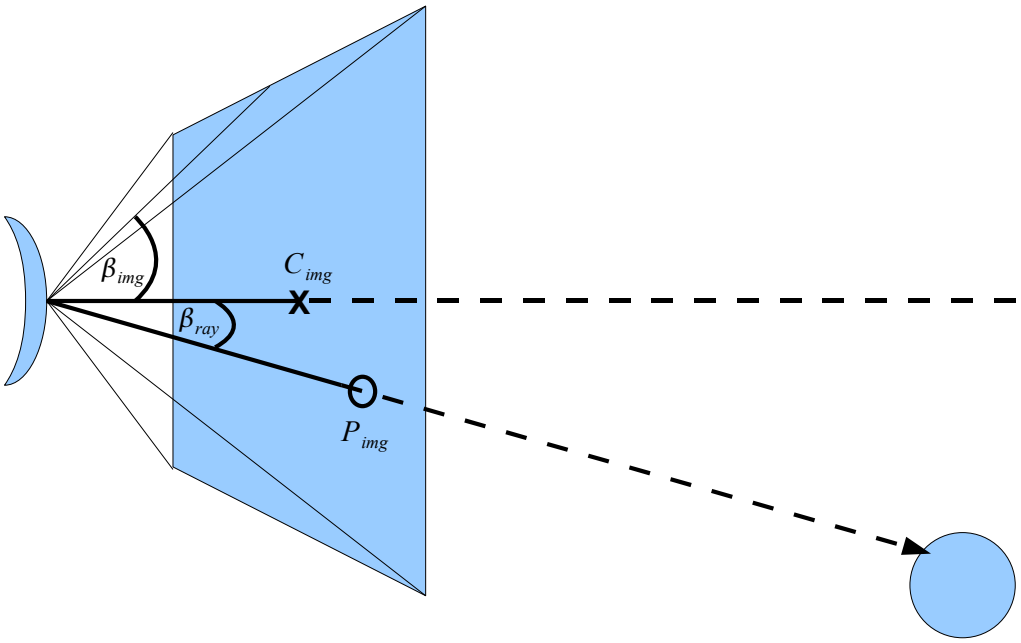


Abbildung 6.1: Darstellung des Sichtstrahls und der dazugehörigen Bildposition.

6.2.1 Circle und Crossing

Die Klassen für Kreise und Kreuzungen sind lediglich Behälter, um die spezifischen Werte bezüglich des Bild- und des Roboterkoordinatensystems zu speichern. Sie werden vom Linien-Erkenner als Zwischenspeicher verwendet, bevor dieser die entsprechenden Perzepte mit den Daten befüllt.

Kreise werden durch ihren Mittelpunkt und den Radius definiert. Außerdem besitzen sie eine bool'sche Variable, die anzeigt, ob diese Werte schon berechnet worden sind.

Die Position einer Kreuzung wird ebenfalls durch ihren Mittelpunkt beschrieben, das heißt, durch den Punkt, an dem sich die schneidenden Linien treffen. Eine Kreuzung setzt sich aus mindestens zwei und maximal vier geraden Linien zusammen. Da aber eine Kreuzung im Gegensatz zu einem Kreis eine Orientierung aufweist, muss diese ebenfalls definiert werden. Hierzu wird willkürlich eine der eingehenden Linien als Bezugspunkt genommen und deren Winkel berechnet. Das Vorhandensein von Kreuzungsteilen wird durch ein bool'sches Array signalisiert, welches (gegen den Uhrzeigersinn, ausgehend von der Bezugslinie) angibt, aus welchen Teilen sich die Kreuzung zusammensetzt – möglich sind Ecken, T- und komplette Kreuzungen. Dieser Aspekt muss bei der späteren Generierung von Kreuzungen durch den Linien-Erkenner berücksichtigt werden.

6.2.2 Line

Da die Instanzen der Klasse `Line` nicht nur als reine Zwischenspeicher Verwendung finden, sondern häufig miteinander verglichen werden müssen, stellen sie bestimmte Mechanismen zur Verfügung, die dem Effizienzgewinn dienen.

Der Zugriff auf sämtliche Instanzvariablen (hauptsächlich Start- und Endpunkt der Strecke) wird über *getter*- und *setter*-Methoden geregelt. Das führt zu der Möglichkeit, festzustellen, wenn sich bestimmte Variablen ändern. Dadurch kann die für Vergleiche oft, aber ansonsten eher selten, benötigte Orientierung einer Linie immer genau so häufig berechnet werden, wie dieses erforderlich ist. Ein *Flag* kennzeichnet, ob die Orientierung der Linie für den aktuellen Anfangs- und Endpunkt schon berechnet wurde. Nur falls dieses nicht der Fall ist und die Orientierung angefordert wird, wird diese auch tatsächlich berechnet.

Gleiches gilt für die Projektion der Linie. Die Endpunkte sind in Bildkoordinaten angegeben. Für deren Projektion in das Roboterkoordinatensystem stellt die Klasse `Line` eigene Methoden (`getProjectedStart` und `getProjectedEnd`) zur Verfügung. Auch hier wird nur eine erneute Berechnung durchgeführt, wenn für die aktuellen Endpunkte noch keine Projektion durchgeführt wurde. Da der Feldlinien-Erkenner Linien sowohl in Bild- als auch in Roboterkoordinaten verarbeitet, können dadurch aufwendige Rechenschritte minimiert werden.

6.2.3 Curve

Die Klasse für Kurven ist wiederum hauptsächlich eine Behälterklasse, die eine Menge beliebiger Punkte enthält. Diese werden als geordnet angenommen, das bedeutet, in irgendeiner Weise besteht eine Verbindung zwischen zwei aufeinander folgenden Punkten. Um diese Ordnung nicht zu stören und eine gewisse Transitivität auszudrücken, können zwar beliebig Punkte innerhalb der Kurve gelöscht, aber nur am Ende neue eingefügt werden.

Diese Klasse wird derzeit ebenfalls vom Feldlinien-Erkenner verwendet. Hier werden in einem ersten Schritt der Konstruktion von Feldlinien, an aufeinander folgenden Stellen die Mittelpunkte einer Linie berechnet und einer Kurve hinzugefügt. Anschließend können dann aus diesen adjazenten Mittelpunkten (das heißt, sie sind über Feldlinien verbunden) gerade Teillinien oder der Mittelkreis generiert werden.

Da Kurven mit vielen Koordinatenpunkten prinzipiell aussagekräftiger sind, werden Vergleichsoperatoren für dieses Kriterium bereitgestellt. Dadurch lassen sich mehrere Kurven quasi nach ihrer Relevanz ordnen.

6.2.4 Run

Die Klasse `Run` ist an das Prinzip des *Run-Length-Encoding* angelehnt. So beschreibt eine `Run`-Instanz einen Teil einer Zeile oder Spalte des Bildes durch Angabe des Start- und des Endpunktes und auch der Farbe, die sich in diesem Bereich nicht ändert. Da es aber keinen Mechanismus gibt, der diese Forderung kontrolliert, ist es dem Benutzer überlassen, inwiefern er sich an dieses Prinzip hält. Auf diese Weise steht allerdings auch einer Rauschunterdrückung nichts im Wege, bei der zum Beispiel einzelne andersfarbige Pixel innerhalb eines Runs zulässig sind.

Wie schon angedeutet gibt es also ausschließlich horizontale und vertikale *Runs*. Für die derzeitige Implementierung der ActiveVision ist das vollkommen ausreichend. Hier werden *Runs* nur im `ActiveLinesPerceptor` eingesetzt, um die Breitenausmaße von Feldlinien zu beschreiben. Aus diesem Grund beinhaltet die Klasse `Run` auch die bool'sche Instanzvariable `increaseNext`, die signalisiert, in welche Richtung ein *Run* zum Aufbau einer Linie (es wären aber auch andere Konstrukte denkbar) erzeugt wurde. Genaueres dazu wird in Abschnitt 6.4.4 beschrieben.

6.2.5 Allgemeine geometrische Methoden

Allgemeine geometrische Funktionen, die von unterschiedlichen Komponenten der ActiveVision benötigt werden, werden in der Datei `ActiveVisionGeometry` über statische Methoden zur Verfügung gestellt. Sie sollen im Folgenden kurz erläutert werden.

closeAngles Diese Funktion existiert in zwei Ausprägungen und verifiziert, ob die Differenz zweier Winkel innerhalb einer vorgegebenen Toleranz liegt. Hierbei wird einmal berücksichtigt, dass Winkeldifferenzen von ungefähr 360° (nachdem die Winkel normalisiert worden sind) auch als ähnlich bewertet werden können. Die andere Variante wendet das gleiche Prinzip für 180° an, was zum Beispiel beim Vergleich von Geraden hilfreich ist, die einfach entgegengesetzt definiert sind.

isLine Hierbei wird für drei übergebene Punkte festgestellt, ob diese auf einer Geraden liegen. Dieses ist genau dann der Fall, wenn die Steigung für eine Gerade zwischen dem ersten und zweiten Punkt und einer zwischen dem ersten und dritten Punkt gleich ist. Benötigt wird dieser Test beispielsweise, wenn aus drei Punkten ein Kreis berechnet werden soll, wozu diese eben nicht auf einer einzelnen Gerade liegen dürfen.

getDistPointLine In dieser Methode wird der minimale Abstand eines Punktes (P) zu einer Geraden (S) berechnet. Dazu benötigt man einen orthogonalen Vektor zur

gegebenen Geraden ($\vec{O}$) und einen Vektor vom gegebenen Punkt zur Geraden ($\vec{PS}$). Dann ergibt sich der Abstand ($dist$) wie in (6.2) gezeigt (siehe hierzu auch [18]). Diese Berechnungen werden häufig bei der Rekonstruktion der Feldlinien verwendet, um zum Beispiel festzustellen, ob zwei durch Linien definierte Geraden ähnlich sind. In diesem Fall können die Feldlinienfragmente nämlich zusammengefasst werden.

$$dist = \frac{|\vec{O}_x \cdot \vec{PS}_x + \vec{O}_y \cdot \vec{PS}_y|}{\|\vec{O}\|} \tag{6.2}$$

isPointOnLine `isPointOnLine` überprüft, ob ein Punkt innerhalb eines gegebenen Bereichs auf beziehungsweise neben einer begrenzten Linie liegt. Hierzu wird erst festgestellt, ob sich der Punkt innerhalb eines Rechteckes befindet, welches durch die Enden der Linie zuzüglich des gegebenen Puffers aufgespannt wird. Anschließend wird der Abstand des Punkts zur theoretisch durch die Linie definierten Geraden gemessen (siehe vorherigen Paragrafen). Überschreitet dieser die gegebene Toleranz nicht und ist auch das erste Kriterium erfüllt, liegt der Punkt mehr oder weniger auf der vorgegebenen Linie. Dadurch kann etwa festgestellt werden, ob der Schnittpunkt zweier durch Linienfragmente beschriebener Geraden auch nahe der beschränkten Linien liegt. Dann wäre eine Kreuzung gefunden.

getIntersection Diese Funktion berechnet den Schnittpunkt zweier Geraden (jeweils definiert durch zwei Punkte) und orientiert sich dabei direkt an [17]. Sie wird benötigt um Kreuzungen von Feldlinien zu bestimmen.

getCenter In dieser Methode wird anhand dreier Punkte der Mittelpunkt des von ihnen definierten Kreises bestimmt. Die Berechnung ist genau dann eindeutig beziehungsweise überhaupt möglich, wenn die drei Punkte nicht auf einer Geraden liegen. Geometrisch gesehen, konstruieren sie dann ein Dreieck, welches genau in den Kreis passt. Der Kreismittelpunkt ergibt sich als der Schnittpunkt der drei Mittelsenkrechten. Diese Funktion ist zum Beispiel hilfreich beim Bestimmen der Ballparameter.

6.3 Bildverarbeitungsmodul

Die Implementierung des Bildverarbeitungsmoduls der ActiveVision befindet sich im Verzeichnis `activevision/imageprocessor`. Der `ActiveImageProcessor` stellt das endgültige Modul dar, welches einem Prozess zugeordnet werden muss, um die Bildverarbeitungsschritte zu initiieren. Er erbt von der abstrakten Klasse `ImageHandler`, welche über das Interface

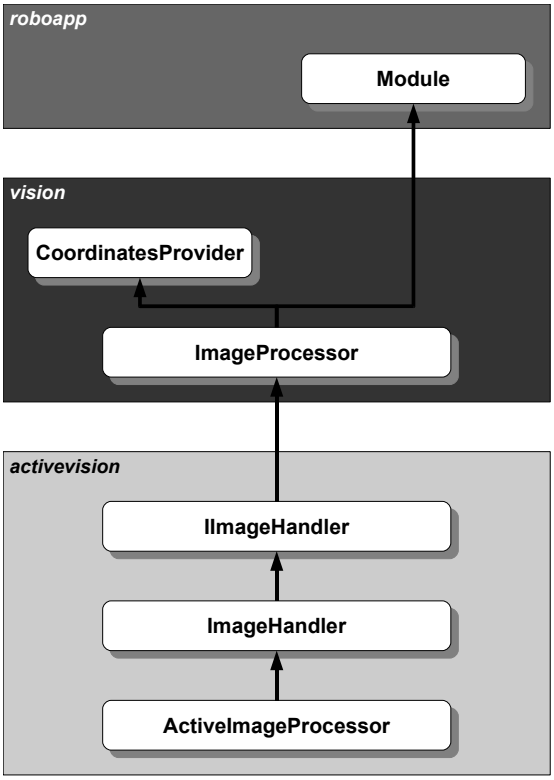


Abbildung 6.2: Vererbungshierarchie der Bildprozessoren.

`IImageHandler` eine Spezialisierung des `ImageProcessor`'s repräsentiert. Der `ImageProcessor` ist eine Implementierung des bisherigen Bildverarbeitungsansatzes (deshalb im Namensraum `vision` zu finden) und wird an dieser Stelle nur kurz beschrieben. Die Zusammenhänge verdeutlicht Abbildung 6.2 noch einmal.

6.3.1 ImageProcessor

Diese Klasse bietet in erster Linie die Funktionalität, Bildkoordinaten in das Koordinatensystem des Roboters zu übersetzen. Damit implementiert sie das Interface `CoordinatesProvider`, welches entsprechende Methoden definiert.

Außerdem erbt der `ImageProcessor` von der Klasse `Module`, wodurch er alle Voraussetzungen erfüllt, um vom *RoboFrame* verwaltet und innerhalb eines Prozesses ausgeführt werden zu können.

Durch die angegebenen Eigenschaften des `ImageProcessor`'s ist es auch dem `ActiveImageProcessor` möglich, reale Positionen zu Bildkoordinaten zu berechnen und innerhalb des Frameworks als Bildverarbeitungsmodul eingesetzt zu werden.

6.3.2 ImageHandler

In diesem Abschnitt wird der `ImageHandler` beschrieben. Dabei wird keine Unterscheidung zu Implementierungen gemacht, die eigentlich schon in der Elternklasse `IImageHandler` stattfinden. Prinzipiell kann man beide Klassen als eine Einheit betrachten. Sinn und Zweck der abstrakten Elternklasse ist lediglich, die Funktionalität der Template-Klasse `ImageHandler` nicht-Template-Klassen zur Verfügung zu stellen. Dadurch können die Perzeptoren, die über diese Klasse auf Bildinformationen zugreifen, ohne Templates realisiert werden. Die konkreten Typen der Kameras, Bilder, Farbtabellen und der eingesetzten Pixelklassen müssen sie daher nicht interessieren. Sie bleiben völlig unabhängig davon und nutzen ausschließlich den `ImageHandler` beziehungsweise eben dessen Interface.

In der Template-Klasse wurden nur die Methoden implementiert, die tatsächlich von den Parametern des Templates abhängen. Die restlichen Implementierungen befinden sich schon in der Elternklasse. Das schafft eine bessere Übersicht, welche Funktionen auf die Template-Parameter angewiesen sind und welche allgemeinerer Natur sind. Im Folgenden wird nun also eine genauere Darstellung der Gesamtfunktionalität des `ImageHandlers` präsentiert.

Insgesamt bietet diese Einheit Methoden,

- die einfache Werte bezüglich des Bildes zurückliefern (zum Beispiel Höhe und Breite)
- die komplexere Werte des Bildes zurückliefern (beispielsweise die Distanz der unteren Bildecken zueinander, nachdem diese in das Roboterkoordinatensystem projiziert worden sind)
- die überprüfen, ob Koordinaten innerhalb des Bildes liegen
- die zu gegebenen Koordinaten die Farbe des entsprechenden Pixels bestimmen
- die in einer bestimmten Richtung nach dem Anfang oder dem Ende einer bestimmten Farbe suchen
- die für zwei Bildpunkte die reale Entfernung berechnen
- die berechnen, wie groß eine reale Distanz an einer bestimmten Stelle im Bild wäre
- die zu einer gegebenen Position in Roboterkoordinaten die entsprechenden Bildkoordinaten bestimmen

Nach dieser Übersicht wird nun im Folgenden auf einzelne Teilaspekte genauer eingegangen.

Einfache Werte des Bildes

Die hier als „einfach" bezeichneten Werte sind zum einen die feste Breite und Höhe des Bildes, zum anderen aber auch ein Wert, der ausdrückt, wie nahe am Bildrand zuverlässige Pixel erwartet werden können. Dieser ist zur Laufzeit durch den Benutzer anpassbar und notwendig, da die korrekte Farbgebung der Pixel am Bildrand abnehmen kann. So werden standardmäßig die zwei äußersten Pixelreihen nicht von der Bildverarbeitung berücksichtigt. Die „einfachen" Werte werden nicht für jedes Bild neu bestimmt, sondern bleiben prinzipiell konstant.

Komplexere Werte des Bildes

Zu den komplexeren Werten des Bildes zählt die reale Distanz zwischen den unteren beiden Bildecken, das heißt, die Entfernung zwischen den projizierten Eckpunkten. Gleiches gilt für die oberen beiden Bildecken. Diese Werte werden einmal pro Bild berechnet und dienen dazu, innerhalb des aktuellen Bildes reale Distanzen durch Interpolation zu bestimmt. Wie die Berechnung dieser Werte einerseits und die Interpolation andererseits vonstattengeht, zeigen die Abschnitte „Berechnung realer Distanzen für Bildkoordinaten (Bild $\Rightarrow$ Realität)" und „Berechnung realer Distanzen innerhalb des aktuellen Bildes (Realität $\Rightarrow$ Bild)".

Farbklassifikation von einzelnen Pixeln

Wie zu einzelnen Bildkoordinaten die Farbklasse des entsprechenden Pixels bestimmt werden kann, soll hier nur kurz gezeigt werden, um den Mechanismus darzustellen, der direkt aus der bisherigen Bildverarbeitung übernommen und in einer eigenen Funktion gekapselt worden ist (siehe Listing 6.1). Hierbei wird eine Abfrage zu Koordinaten außerhalb des Bildes abgefangen und eine Fehlermeldung ausgegeben, aber ein Absturz der Anwendung verhindert. Die Klassifikation findet in drei Schritten statt. Als erstes muss das Roh-Pixel an den gegebenen Koordinaten aus dem Bild extrahiert werden. Danach wird dieses mittels Farbtabelle in eine Repräsentation übersetzt, die die zugehörige Farbklasse beinhaltet. Zum Schluss kann daraus die Farbklasse des Pixels an der geforderten Stelle bestimmt werden.

Suche des Anfangs einer bestimmten Farbe

Der `ImageHandler` stellt Methoden zur Verfügung, mit denen von einem Punkt aus in verschiedene Richtungen nach dem Beginn einer bestimmten Farbe gesucht werden kann. Dabei können zusätzlich Parameter übergeben werden, die die Schrittweite bei

```
const unsigned short getColor(
  const int x,
  const int y) const
{
  if (x >= 0 && x < imageSize.x
    && y >= 0 && y < imageSize.y) {
      const ConcretePixelIn inPixel
        = image->getPixel(x, y);
      const ConcreteClassifiedPixel& pixel
        = colorTable.lookup(inPixel);
    return pixel.getColorClass();
  }
  else {  // out of image
    LOGPATH_ERROR("ImageHandler.getColorClass()",
      "(%i,%i) out of image", x, y);
    return common::math::MAX_SHORT;
  }
}
```

Listing 6.1: Methode zur Bestimmung von Farbklassen (kodiert als positive ganze Zahlen) zu gegebenen Bildkoordinaten (x, y)

der Suche bestimmen, und wieviele Pixel am Stück gefunden werden müssen, um Bildfehler auszuschließen.

Dadurch kann die Suche an unterschiedliche Gegebenheiten angepasst werden. Wird zum Beispiel festgestellt, dass die aufgenommenen Bilder sehr fehlerhaft sind, kann der Wert für benötigte Pixel vom Anwender erhöht werden. Dieses würde Fehlidentifikationen von Objekten minimieren. Sucht der Linien-Erkenner beispielsweise nach dem Anfang einer Feldlinie, kann die Schrittweite an die Linienbreite angepasst werden. Dadurch wird die Effizienz der Suche erhöht, indem größere Abstände der analysierten Punkte gewählt werden können, ohne Feldlinien zu „überspringen".

Die Suche selbst läuft dann folgendermaßen ab. Die Koordinaten des Ausgangspunktes werden in einer Schleife um die definierte Differenz in gegebener Richtung verändert, bis der Bildrand (oder die vorgegebene Schranke) erreicht oder eine passende Farbregion gefunden wird. In jedem Schleifendurchlauf wird das den aktuellen Koordinaten entsprechende Pixel klassifiziert. Ist es das erste einer passenden Farbsequenz, wird es abgespeichert. Falls genügend Pixel der gleichen Farbe folgen, wird die gemerkte Position zurückgeliefert. Andernfalls wird die Suche fortgesetzt.

Suche nach dem Ende einer bestimmten Farbe

Die Suche nach dem Farbende gestaltet sich ähnlich des Auffindens von Farbanfängen, nur das diesmal in eine spezifizierte Richtung eben nach dem Ende der Farbe des Ausgangspixels gesucht wird. Wieder kann die Schrittweite angepasst und zusätzlich

ein Wert angegeben werden, der diesmal bestimmt, wie viele Pixel einer anderen Farbe noch als Rauscheffekte betrachtet werden.

In einer Schleife wird über die Pixel in die festgelegte Richtung iteriert. Solange Pixel der gleichen Farbe identifiziert werden und der Bildrand nicht erreicht ist, wird das nächste Pixel entsprechend der Schrittweite ausgewählt.

Würde das nächste Pixel außerhalb des Bildes liegen, wird die Schrittweite solange halbiert, bis es innerhalb des Bildes läge oder die Schrittweite Null wird – der Bildrand also erreicht ist. In diesem Fall ist das Ende der Farbe dort gefunden worden und die Iteration kann abgebrochen werden.

Prinzipiell ähnlich ist das Vorgehen, wenn ein Pixel einer anderen Farbe gefunden wird. Auch hier wird die Schrittweite solange halbiert bis das nächste Pixel die richtige Farbe aufweist oder eine minimale Schrittweite erreicht wurde. Bei erneut passender Farbe wird die Suche mit der ursprünglichen Schrittweite fortgesetzt. Ansonsten werden mit minimaler Schrittweite die folgenden Pixel untersucht, bis entweder doch wieder korrekte Pixel gefunden werden oder die Grenze für andersfarbene Pixel überschritten wird. Im ersten Fall kann die Suche wieder mit voller Schrittlänge aufgenommen werden. Im zweiten Fall wurde das Ende mit dem letzten Punkt der geforderten Farbe gefunden und die Suche wird beendet.

Wie unschwer zu erkennen ist, basiert die angewendete Strategie (sowohl hier als auch im vorherigen Abschnitt) also auf einer Art binären Suche, indem die Schrittlängen halbiert werden, wenn andersfarbene Pixel erkannt wurden. Das verspricht einen sehr effizienten Ansatz, da erst relativ große Schritte gemacht werden können, und dann eine binäre Suche eingesetzt wird, wenn zwischen zwei Schritten der Farbrand liegt.

Berechnung realer Distanzen für Bildkoordinaten (Bild $\Rightarrow$ Realität)

Diese Funktion ist schnell beschrieben. Gegeben sind zwei Bildkoordinaten, deren Abstand zueinander in der Realität bestimmt werden soll. Dazu werden beide Punkte einfach auf den Boden bezüglich des Roboterkoordinatesystems (xy-Ebene) projiziert, und deren Entfernung kann mittels gewöhnlicher Abstandsberechnungen im zweidimensionalen Raum kalkuliert werden.

Ein Nachteil dieses Vorgehens ist allerdings, dass der berechnete Wert nicht korrekt ist, wenn mindestens einer der beiden Punkte nicht dem Untergrund zuzuordnen ist, sondern einem Objekt, dass sich vom Boden abhebt. Diesem Missstand ist jedoch nicht ohne weiteres beizukommen und zumindest für den Einsatz in der ActiveVision sind keine gravierenden Einschränkungen zu erwarten. Da sie mehr oder weniger den Boden nach relevanten Informationen absucht (da sich alle Objekte und Feldlinien darauf befinden), interessiert hier ohnehin eher die Entfernung zweier Punkte in Bezug auf

den Untergrund.

Berechnung realer Distanzen innerhalb des aktuellen Bildes (Realität ⇒ Bild)

Wie schon angedeutet, werden reale Entfernungen innerhalb des Bildes ausschließlich interpoliert. Allerdings tritt dasselbe Problem wie im vorherigen Abschnitt auf, dass die berechneten Entfernungen sich nur auf den Boden beziehen. Aber auch hier ist es für die Komponenten der ActiveVision genauso unproblematisch.

Sodann werden Distanzen wie folgt bestimmt. Zuerst wird berechnet, welche Ausmaße die reale Distanz am oberen und unteren Bildrand hätte, indem sie mit den bekannten realen Längen der Bildränder (siehe „Komplexere Werte des Bildes") verglichen wird. Danach kann aus den resultierenden beiden Werten für eine Zeile innerhalb des Bildes der zugehörige Wert der Distanz interpoliert werden. Im Folgenden wird die Berechnungsformel noch einmal dargestellt:

$$d_{img}(y) = d_{img_{min}} + \frac{y}{h} \cdot (d_{img_{max}} - d_{img_{min}}) \tag{6.3}$$

wobei:	$d_{img_{min}}$	Länge der realen Distanz des oberen Bildrands
	$d_{img_{max}}$	Länge der realen Distanz des unteren Bildrands
	h	Höhe des gesamten Bildes
	y	Zeile im Bild, in der Ausmaß der realen Distanz gesucht wird

Als Beispiel: Wird innerhalb einer Zeile nach dem Ball gesucht, kann man bestimmen, welche Länge des Ballradius' zu erwarten ist, wenn der Ball tatsächlich in der aktuellen Zeile zu sehen wäre. Die Suchschritte in dieser Zeile könnten daran angepasst werden.

Diese einfache Form zur Bestimmung von Längenmaßen funktioniert allerdings nur solange, wie die y-Achse der Bilder nahezu parallel zum Boden steht. Dieses ist bei der momentanen Konstruktion des Roboters der Fall und wird wohl auf unbestimmte Zeit so beibehalten, da es keine Gründe gibt, die dagegen sprechen. Es wäre aber auch kein Problem die Methode anzupassen, allerdings orientiert sich die aktuelle Implementierung eben stark an der Forderung nach möglichst performantem Verhalten, und die Genauigkeit genügt den Ansprüchen der ActiveVision-Komponenten.

Projektion realer Positionen in das aktuelle Bild

Ein wesentlicher Bestandteil der ActiveVision ist, Punkte, die in Roboterkoordinaten gegeben sind, als Ausgangskoordinaten für die Analyse des aktuellen Bildes zu verwenden. Dazu müssen diese Ausgangspunkte aber erst in Bildkoordinaten übersetzt

werden, und genau dieses leistet die Methode `getImagePosition`. Sie geht wie folgt vor.

Als erstes wird die Transformationsmatrix vom Kamera- zum Roboterkoordinatensystem berechnet. Wendet man diese auf den dreidimensionalen Vektor an, der die reale Position beschreibt, so erhält man den Sichtstrahl von der Kamera zu eben diesem Punkt. Mit dem ermittelten Sichtstrahl kann dann anhand einer Funktion der Kamera-Klassen (siehe Abschnitt 6.1), die entsprechende Position im Bild bestimmt werden.

6.3.3 ActiveImageProcessor

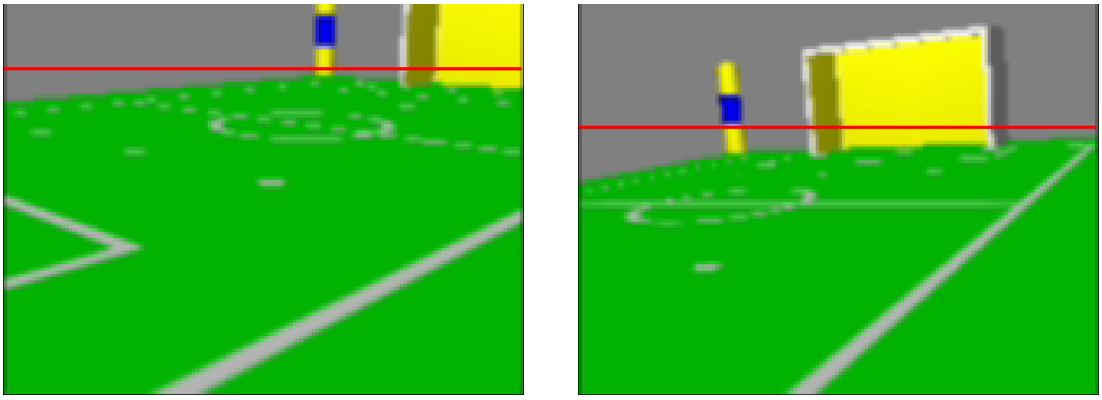
Im eigentlichen Bildverarbeitungsmodul der ActiveVision werden die einzelnen Perzeptoren verwaltet und ausgeführt. Es ist als Template realisiert und erbt direkt vom `ImageHandler`. Die Perzeptoren werden in einem Vektor abgelegt und immer in der gleichen Reihenfolge ausgeführt.

Bei der Initialisierung des Moduls (zu diesem Zeitpunkt sind bereits alle Perzeptoren registriert) werden alle relevanten Parameter aus einem sogenannten `EditableVariantSet` gelesen und an die entsprechenden Komponenten weitergegeben. Durch diese Implementierung wird die gewünschte hohe Flexibilität erreicht, welche erlaubt, einzelne Parameter auch noch zur Laufzeit anzupassen. Außerdem wird bei den Erkennern ein Verweis auf das Modul als `ImageHandler` gespeichert, damit diese die bereitgestellten Funktionen bezüglich der Bildverarbeitung nutzen können.

Wird der `ImageProcessor` durch den übergeordneten Prozess aufgerufen (`execute`), finden als erstes Statusabfragen statt, mit denen bestimmt werden kann, ob eine momentane Ausführung tatsächlich sinnvoll ist. Mögliche Zustände, in denen keine Bildanalyse durchgeführt wird, sind (äquivalent zur bisherigen Bildverarbeitung):

- Es ist kein neues Bild vorhanden
- Es liegt keine neue Kameramatrix vor
- Der Roboter ist umgefallen

Kommt es doch zur Ausführung, wird das aktuelle Bild aus dem entsprechenden Puffer geladen. Anschließend wird eine Horizont-ähnliche Grenze im Bild berechnet (siehe Abbildung 6.3), die eine vordefinierte Entfernung zum Roboter darstellt, ab der in der Regel eine Bildverarbeitung keinen Sinn mehr macht. Standardmäßig ist sie so gewählt, dass sie der Spielfelddiagonalen plus einem Puffer entspricht. Diese Begrenzung wird dann an die einzelnen Perzeptoren übergeben, die sie aber auch ignorieren dürfen.



(a) (b)

Abbildung 6.3: Beispiele der berechneten oberen Verarbeitungsgrenze in unterschiedlichen Bildern aus dem Simulator. Diese Grenze kennzeichnet standardmäßig ungefähr einen Abstand zum Roboter, der etwas länger als die Spielfelddiagonale ist. Man kann erkennen, dass in keinem Fall fälschlicherweise ein Teil des Feldes nicht berücksichtigt würde (also oberhalb der Schranke läge). Bei den Aufnahmen schaute der Roboter über die gesamte Diagonale des Feldes. Bei (a) war die Kamera nicht um die nach oben gerichtete Achse rotiert und in (b) maximal (100°).

Im folgenden Schritt werden die realen Längen der oberen und unteren Bildränder (nach deren Projektion auf den Boden) bestimmt, um später Entfernungen innerhalb des Bildes effizient abschätzen zu können (siehe Abschnitt 6.3.2). Zum Schluss werden die einzelnen Erkenner zur Ausführung gebracht. Dabei werden nur diejenigen aufgerufen, deren Perzepte tatsächlich von anderen Modulen angefordert werden.

6.4 Perzeptoren

Die Perzeptoren genieren unterschiedliche Perzepte und leisten die eigentliche Erkennungsarbeit, weshalb sie in dieser Arbeit auch als Erkenner bezeichnet werden. Perzepte beschreiben die für einzelne Objekte (oder Feldlinien – die etwas gesondert betrachtet werden) die erkannten relevanten Daten (zum Beispiel die Position des Balls). Anschließende Module der *RoboCup*-Applikation verwenden dann diese Perzeptdaten, um beispielsweise die Ballmodellierung mit neuen Werten zu versorgen.

Die Vererbungshierarchie der Perzeptoren ist wie folgt aufgebaut und wird in Abbildung 6.4 noch einmal verdeutlicht. Im Gegensatz zur Hierarchie der Bildprozessoren (siehe Abbildung 6.2) ist sie komplett unabhängig von der bisherigen Implementierung der Bildverarbeitung .

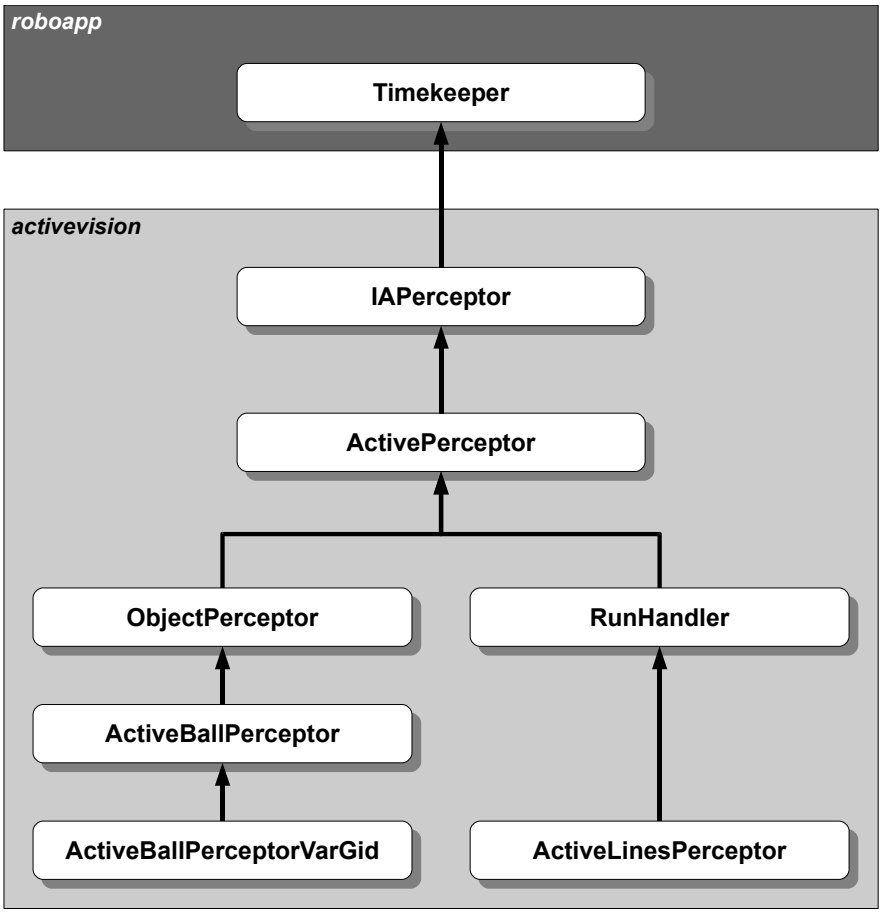


Abbildung 6.4: Vererbungshierarchie der Perzeptoren.

Basisklasse ist der `IAPerceptor`. Er stellt sicher, dass die Perzeptoren alle Methoden zur Verfügung stellen, die für die allgemeine Verwaltung notwendig sind (beispielsweise die Bereitstellung von Informationen über die verwendeten Puffer). Durch die Framework-spezifische Elternklasse `Timekeeper` können Funktionen zur Zeitmessung bereitgestellt werden.

Der `ActivePerceptor` definiert Funktionen, die vom aufrufenden Bildprozessor zur Perzeptorinitialisierung oder -ausführung benötigt werden. Er stellt die letzte Klasse der Hierarchie dar, von der alle Perzeptorimplementierungen erben müssen. Wird das bis zu dieser Stelle definierte Interface vollständig realisiert, kann ein solcher Perzeptor vom `ActiveImageProcessor` eingestetzt werden.

Schon konkretere aber immernoch abstrakte Klassen stellen der `ObjectPerceptor` und der `RunHandler` dar. Sie stellen Methoden bereit, die zum einen die Suche nach einzelnen Objekten – Farbregionen aber auch komplexere Strukturen innerhalb des Bildes – und zum anderen das Auffinden und Verwalten von Farb-*Runs* ermöglichen.

Letztendlich bestehen derzeit drei einsatzfähige Perzeptoren – zwei zur Ball- und einer zur Linienerkennung. Die Ball-Perzeptoren erben von der Klasse `ObjectPerceptor`, wie es für Erkenner von Gegenständen im weiteren Sinn vorgesehen ist, und der Feldlinien-Erkenner als ein Sonderfall nur vom `RunHandler`. Es wäre aber auch vorstellbar, dass ein Perzeptor von beiden Klassen erbt (zum Beispiel ein Linien-Perzeptor, der aber einzelne Linien ausgehend von einer vorher bekannten Startposition sucht).

Im folgenden wird auf die genauere Darstellung des `IAPerceptor`'s und des `ActivePerceptor`'s verzichtet und sich auf die restlichen Umsetzungen konzentriert, da diese die tatsächlichen Innovationen enthalten.

6.4.1 Allgemeiner Objekt-Perzeptor

Die wesentlichen Funktionen, die der `ObjectPerceptor` zur Verfügung stellt, beinhalten die Suche nach einer bestimmten Farbregion ausgehend von einem Startpunkt. Hier gibt es zwei Varianten, die sich dadurch unterscheiden, dass der einen ein fixes Suchraster zugrunde liegt und die andere Suchpunkte flexibler anordnet.

Im Folgenden wird die Variante mit fixem Raster beschrieben. Übergabeparameter dabei sind:

- die Farbklasse der gesuchten Region
- die Startposition der Suche
- zwei Werte, die beschreiben in welchem Rahmen um ein Pixel wieviele Pixel

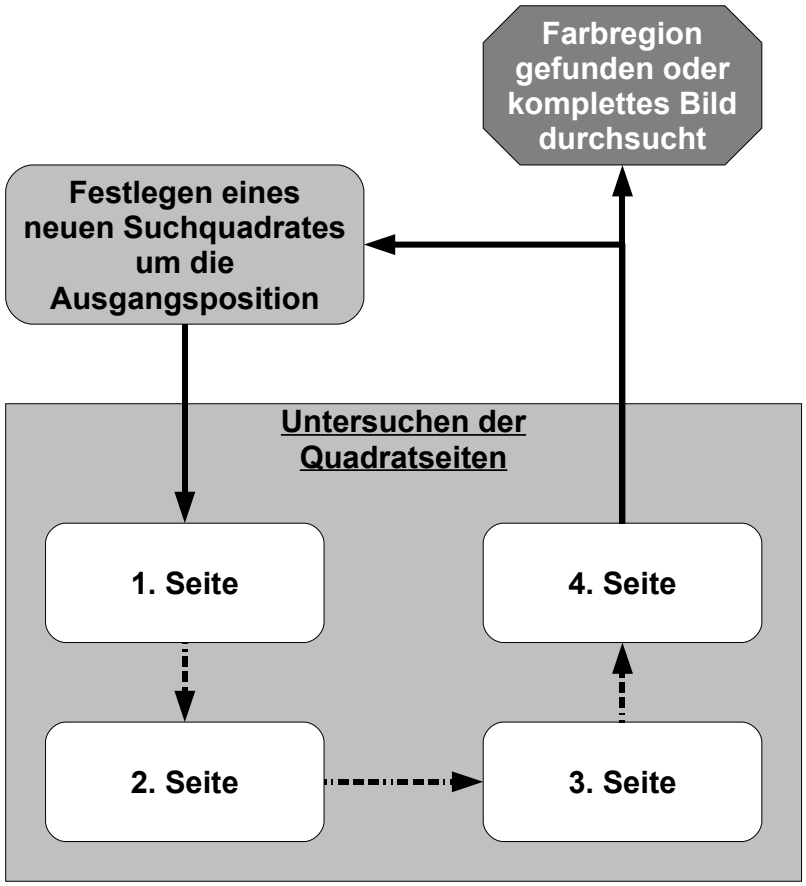


Abbildung 6.5: Zustandsautomat zur Suche nach Farbregionen ausgehend von einem vorgegebenen Startpunkt.

der gleichen Farbe vorhanden sein müssen, damit die Region als gefunden gilt

- der Abstand zwischen den einzelnen Suchpunkten

Die Implementierung orientiert sich an der Umsetzung eines Zustandsautomaten , dessen Zustände in einer Schleife iteriert werden (vereinfacht dargestellt in Abbildung 6.5). Die Suche hält sich dabei an eine für den aktuellen Erkenner gesetzten oberen Schranke im Bild, oberhalb derer keine Pixel untersucht werden (vergleiche Abschnitt 6.3.3).

Im einzelnen ist die Strategie wie folgt angelegt. Im ersten Zustand werden Schranken festgelegt, die ein Quadrat um die Ausgangsposition definieren, die entweder schon innerhalb des Bildes liegt oder andernfalls an die Position verschoben wird, die ihr im Bild am nächsten ist. Das nächst größere Quadrat hat immer um die zweifache Schrittlänge verlängerte Kanten als das vorherige. Allerdings geht diese Rechnung

nicht mehr auf, sobald der Bildrand (oder die vorgegebene absolute ober Schranke) erreicht wird. Dann wird in der ersten Iteration die Begrenzung so angepasst, dass der Bildrand noch untersucht wird. Liegt irgendwann kein Teil dieses Quadrates mehr innerhalb des Bildes beziehungsweise unterhalb der Verarbeitungsbegrenzung, muss die Suche erfolglos abgebrochen werden, da dann bereits das komplette Bild durchsucht worden ist (denkbar wären hier auch Kriterien, so dass nicht das gesamte Bild durchsucht wird, wie es auch schon nach oben hin realisiert ist).

Je nachdem welche Teile des Quadrates noch innerhalb des Bildes liegen, wird jeweils der nächste Zustand gewählt beziehungsweise Zustände übersprungen. Die folgenden Zustände untersuchen die Pixel der Quadratkanten. Die prinzipielle Reihenfolge ist immer gleich und sieht folgende Ordnung vor (die in der zweiten Variante dieser Methode noch optimiert wurde):

1. rechte Kante von oben nach unten
2. untere Kante von rechts nach links
3. linke Kante von unten nach oben
4. obere Kante von links nach rechts

Falls auf einer der Kanten (oder direkt im Ausgangspunkt) ein Pixel der gesuchten Farbe identifiziert wird, findet eine Überprüfung der Punkte innerhalb der definierten Nachbarschaft statt. Ist hier eine ausreichend hohe Zahl an gleichfarbigen Punkten vorhanden, gilt die entsprechende Farbregion als gefunden und die Koordinaten des Pixels werden als Rückgabewert gesetzt.

Die angesprochene zweite Variante der Suchstrategie geht hauptsächlich gleich vor, unterscheidet sich aber darin, dass die Schrittlängen zwischen den einzelnen Suchpunkten variieren. Aus diesem Grund wird auch keine eindeutige Schrittlänge als Parameter übergeben, sondern eine maximale (für den unteren Bildrand) und eine minimale (für die obere Suchgrenze). Der Abstand von Suchpunkten innerhalb des Bildes wird dann anhand dieser beiden Werte interpoliert. Trotzdem wird weiterhin ein Quadrat um die Ausgangsposition aufgespannt und kein Trapez oder ähnliches, was eventuell als sinnvoll erscheinen mag. Tatsächlich erhielte man dadurch aber keinen echten Vorteil gegenüber einem leichter zu ermittelnden und zu „wartenden" Quadrat, wenn die einzelnen Abstände der Punkte auf den Quadratkanten sich sehr wohl an die positionsbezogenen Abstände halten (das heißt, dass benachbarte Punkte eines einzigen Quadrates unterschiedliche Abständen zueinander aufweisen können).

Außerdem hat die Version mit flexiblem Suchraster eine weitere Anpassung erhalten, die dem Einsatz bei der Suche nach ganzen Objekten, anstatt lediglich nach plumpen Farbregionen, entgegen kommt. Sobald ein Pixel der entsprechenden Farbe gefunden wird, kann eine Perzeptor-spezifische Berechnung der Perzeptdaten vorgenommen werden. Schlägt diese fehl (wurde das eigentlich gesuchte Objekt nicht erkannt), kann

direkt weiter gesucht werden. In der ersten Variante müsste eine neue Suche initiiert werden, wenn zwar eine passende Farbregion vorliegt, aber nicht das gewünschte Objekt selbst. Somit ist die zweite Variante zu empfehlen, wenn nicht nur nach einer Farbregion sondern nach einem Objekt gesucht werden soll.

Schlussendlich sei noch auf die erwähnte optimierte Reihenfolge des Pixeltraversierens hingewiesen, bei der nun zuerst die untere Kante, dann die seitlichen Kanten von unten nach oben und zuletzt die obere Kante des Quadrates untersucht werden. Als optimiert gilt diese Strategie, da zum einen erwartet wird, dass relevante Objekte nahe des Roboters lokalisiert sind, und zum anderen eine schnelle Reaktion um so wichtiger ist, je kleiner der Abstand zum gesuchten Objekt ist (zum Beispiel zur Hinderniserkennung oder beim „Ball-Handling").

6.4.2 Klasse zur Generierung und Verwaltung von Runs

Runs werden derzeit ausschließlich vom Feldlinien-Erkenner verwendet, der aus diesem Grund vom `RunHandler` erbt. Die in dieser Arbeit gebrauchte Definition von *Runs* ist Abschnitt 6.2.4 zu entnehmen. Der `RunHandler` implementiert unter anderem die im Folgenden erläuterten Funktionen zum Umgang mit *Runs*.

Auffinden von Runs am Bildrand

Besonders wichtig ist diese Funktion für den `ActiveLinesPerceptor` (siehe Abschnitt 6.4.4). Sie findet *Runs* einer bestimmten Farbe am Bildrand, indem jeweils eine Zeile beziehungsweise Spalte der vier Ränder sukzessive auf Sequenzen von Pixeln entsprechender Farbe analysiert wird. Anschließend werden die gefundenen *Runs* nach Möglichkeit zusammengefasst oder gelöscht, so dass ausschließlich solche übrig bleiben, die zum einen lang genug sind und zum anderen einen Mindestabstand zum nächsten *Run* aufweisen (siehe Abbildung 6.6). In diesem Schritt werden zusätzlich *Runs*, die von einem Bildrand zum anderen reichen, gesondert gemerkt. Dieses ist besonders für den Linien-Perzeptor von Vorteil, da ein solcher *Run* ein starkes Indiz auf eine Feldlinie sein kann.

Generierung einer Kurve ausgehend von einem Run

Die Funktion `expandRun` ist recht komplex und verarbeitet die folgenden Parameter: eine Menge von *Runs* sowie der eigentliche Ausgangs-*Run* plus dessen Vorgänger-*Run*, eine Menge von Linien im Bild, eine Sammlung von Enden bisheriger Kurven und zwei Werte, die zur Bewertung der *Run*-Qualität und zum Vergleich einzelner *Runs* verwendet werden. Im folgenden wird versucht die gesamte Vorgehensweise bei der

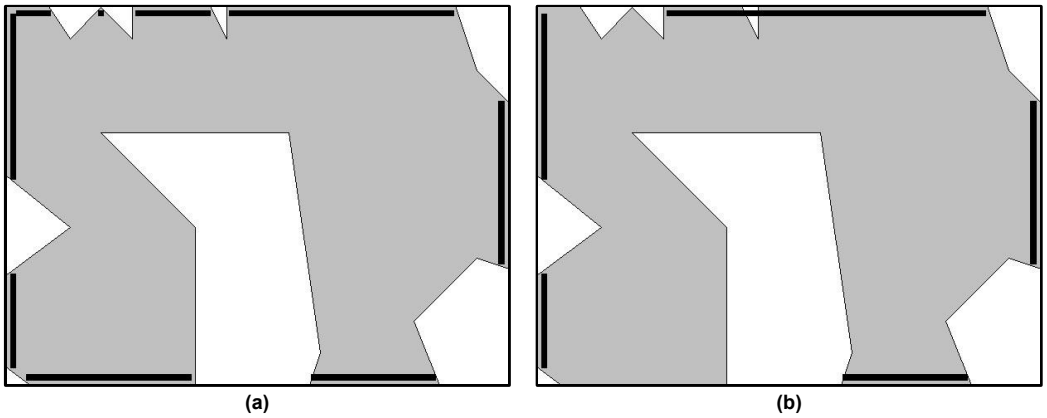


Abbildung 6.6: (a) zeigt alle *Runs* bezüglich des eingezeichneten Objektes, die in einem ersten Schritt am Bildrand gefunden werden. In (b) wurden dann einige *Runs* zusammengefasst oder gelöscht.

Generierung einer Kurve zu erklären, die sich allerdings eigentlich über mehrere Methoden verteilt und schon stark an die Anforderungen des Linien-Erkenners angepasst ist.

Ausgabe der Funktion ist einerseits die generierte Kurve andererseits aber auch neue Anfänge für Kurven, wenn sich die eigentliche Kurve aufspaltet, das heißt, das Kurvengebilde (in der Regel Feldlinien) teilt sich auf (beispielsweise an Linienkreuzungen). In diesem Fall wird die Kurve für eine Abspaltung weiter konstruiert und die anderen als neue Kurvenanfänge abgespeichert, die in späteren Iterationen ausgebaut werden müssten. Die einzelnen Punkte der Kurve ergeben sich aus den Mittelpunkten von *Runs*, die bestimmte Zuverlässigekeitskriterien erfüllen müssen, wie zum Beispiel, dass ihre Länge innerhalb bestimmter Vorgaben liegt.

Damit ist die Aufgabe darauf beschränkt, ausgehend von einem *Run* eine Sequenz von *Runs* zu finden, die mehr oder weniger nebeneinander im Bild lokalisiert sind und die gleiche Farbe beschreiben. Es wird also eine Schleife durchlaufen, die für den aktuellen *Run* die nächsten *Runs* bestimmt. Dieses geschieht in die durch den aktuellen *Run* selbst definierte Richtung (siehe Abschnitt 6.2.4) mit einer Schrittlänge, die allerdings in einem realen Maß vorgegeben ist und in Bildmaße umgerechnet werden muss (siehe Abschnitt 6.3.2). So kann beispielsweise vom `ActiveLinesPerceptor` eine Schrittlänge gewählt werden, die immer an die Linienbreite im aktuellen Bild angepasst ist. Im Detail wird also orthogonal zum aktuellen *Run* ein Schritt gemacht, und an dieser Position parallel zum *Run* innerhalb einer gewissen Reichweite alle *Runs* bestimmt. Wird allerdings festgestellt, dass erzeugte *Runs* eine bestimmte Länge überschreiten, werden an den beiden Enden des aktuellen *Runs* neue generiert, deren Ausrichtung um 90° gedreht ist. Diese dienen dann ihrerseits wiederum als Ausgangspunk-

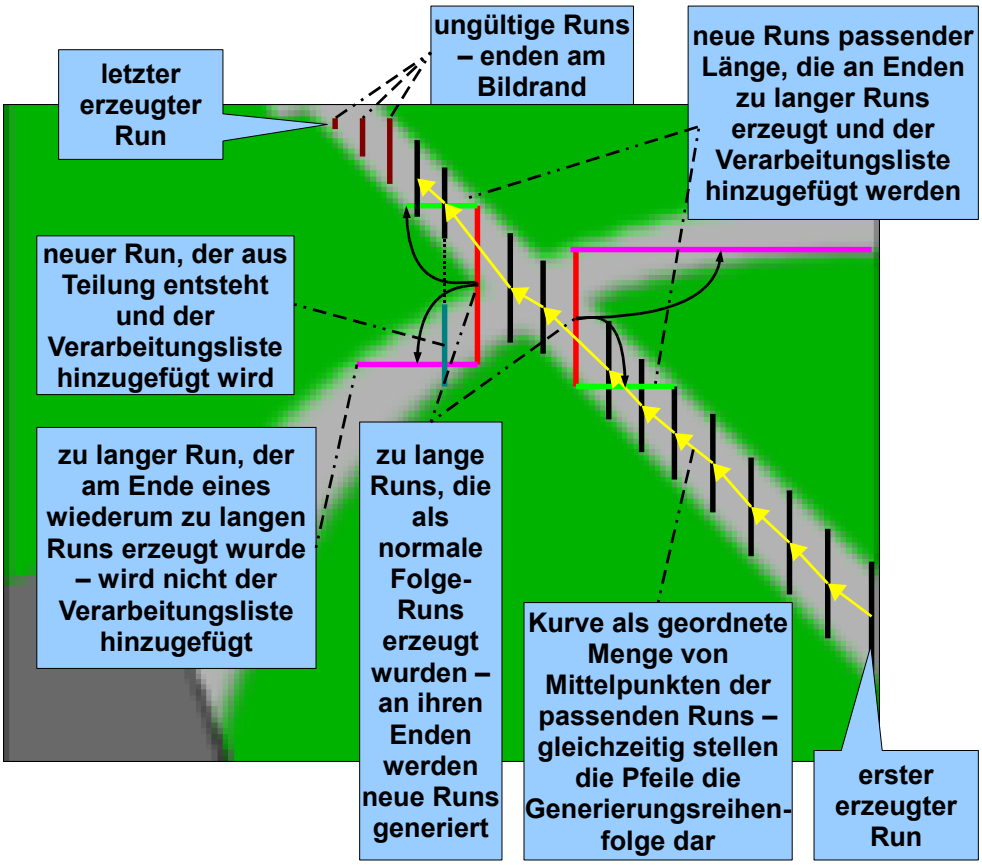


Abbildung 6.7: Vorgehensweise beim Erstellen einer Kurve mittels der Generierung von Feldlinien-*Runs*.

te für Kurven. Die *Runs*, die zur Generierung einer einzelnen Kurve verwendet werden haben immer die gleiche Orientierung (also entweder horizontal oder vertikal).

Die Schleife zur Konstruktion einer Kurve wird dann unterbrochen, wenn ein neu erzeugter nächster *Run* eine schon vorhandene Linie schneidet – zur Verallgemeinerung gegenüber dem Linien-Erkenner können einfach keine Linien oder welche für Bereiche übergeben werden, die tabu sein sollen. Im Rahmen des Linien-Erkenners kann diese Vorgabe dazu genutzt werden, möglichst keine Kurven für Feldliniensegmente zu erzeugen, die schon verarbeitet worden sind. Weitere Abbruchkriterien existieren, falls eine bestimmte Anzahl von *Runs* hintereinander die Zuverlässigkeitsprüfungen nicht bestehen, die sich auf Länge und umgebende Farben beziehen können. Außerdem muss natürlich die Kurvenkonstruktion beendet werden, wenn in der gesuchten Richtung überhaupt keine *Runs* der entsprechenden Farbe mehr vorhanden sind.

Die Vorgehensweise soll anhand von Abbildung 6.7 noch einmal verdeutlicht werden. Abbildung 6.11 stellt die im Bild identifizierten Feldlinien dar.

6.4.3 Ball-Perzeptor

Mit den beiden Ball-Perzeptoren stehen zwei einsatzfähige Prototypen für die Implementierung von Objekt-Erkennern zur Verfügung. Als erstes wird die Vorgehensweise des `ActiveBallPerceptor`'s beschrieben und im Anschluss die Modifikationen, die im `ActiveBallPerceptorVarGrid` vorgenommen wurden.

Das Vorgehen kann in drei Schritte unterteilt werden:

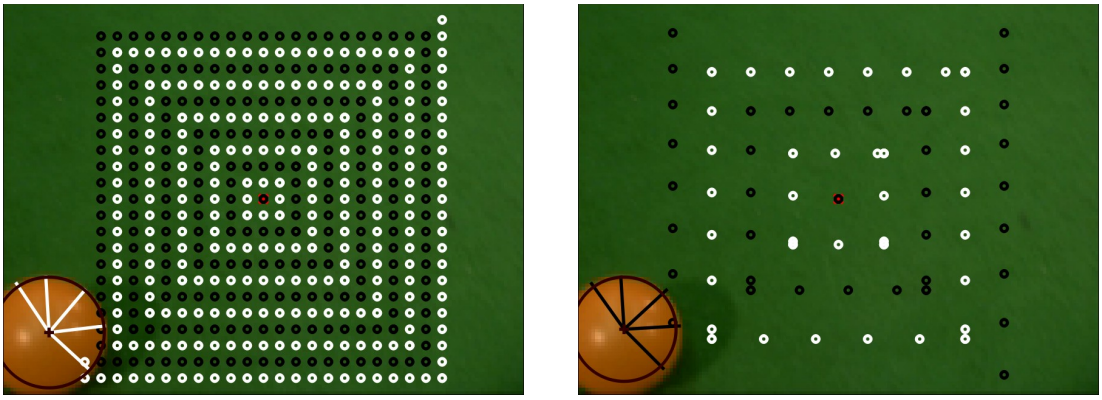
1. Bestimme einen Startpunkt der Suche innerhalb des Bildes.
2. Finde eine Region in der Farbe des Balls (Orange).
3. Berechne die Ballparameter und stelle sie den anschließenden Komponenten zur Verfügung.

Für den ersten Schritt kann auf die Daten des Ballmodells zurückgegriffen werden, die eine Schätzung der Ballkoordinaten zum aktuellen Zeitpunkt liefern. Bei einer ausreichenden Zuverlässigkeitsangabe vom Ballmodell zur Schätzung wird die Ballposition in Bildkoordinaten umgerechnet (siehe Abschnitt 6.3.2). Andernfalls wird der Ausgangspunkt der Suche in die Mitte des Bildes gesetzt.

Danach wird vom Startpunkt aus mittels eines fixen Rasters nach einer Region orangener Farbe gesucht. Diese Funktion ist im `ObjectPerceptor` realisiert (siehe Abschnitt 6.4.1). Falls eine zutreffende Position im Bild gefunden wurde, werden ausgehende von dieser versucht die Parameter des Balls zu bestimmen. Das bedeutet insbesondere, dass die Suche als beendet gilt, sobald eine ausreichend konsistente orangefarbene Region ermittelt worden ist.

Um die Ausmaße des Balls im Bild feststellen zu können, werden lediglich drei Punkte auf dessen Rand benötigt, da dadurch sein Umfang und Mittelpunkt eindeutig bestimmbar sind. Die Schwierigkeit liegt darin, drei Punkte zu finden, die tatsächlich genau am Rand des Balls liegen. Dazu wird vom Startpunkt innerhalb der orangenen Region in genau acht Richtungen nach echten Randpunkten gesucht. Die Suche nach den Randpunkte ist so organisiert, dass jeweils die äußersten Punkte in der entsprechenden Richtungen entdeckt werden. Sie verwendet die Methoden des `ImageHandler`'s um Farbenden zu finden (siehe Abschnitt 6.3.2). Die acht Richtungen sind links, rechts, oben, unten und jeweils Zwischenstufen. Als echte Randpunkte gelten Pixel, die selbst noch orangefarben sind, aber auch an Farbregionen grenzen, die grün oder weiß sind, da diese nur zum Untergrund oder Feldlinien gehören können, das heißt, hier liegt keine Verdeckung des Balls vor.

Letztendlich werden für alle Tripel von Ball-Randpunkten der entsprechende Mittelpunkt und Radius berechnet, welche erst einer Plausibilitätsprüfung unterzogen und dann gemittelt werden. Danach werden die Parameter des Balls in das Roboterkoordinatensystem projiziert (siehe Abschnitt 6.3.1). Dabei wird der berechnete Radius



(a) (b)

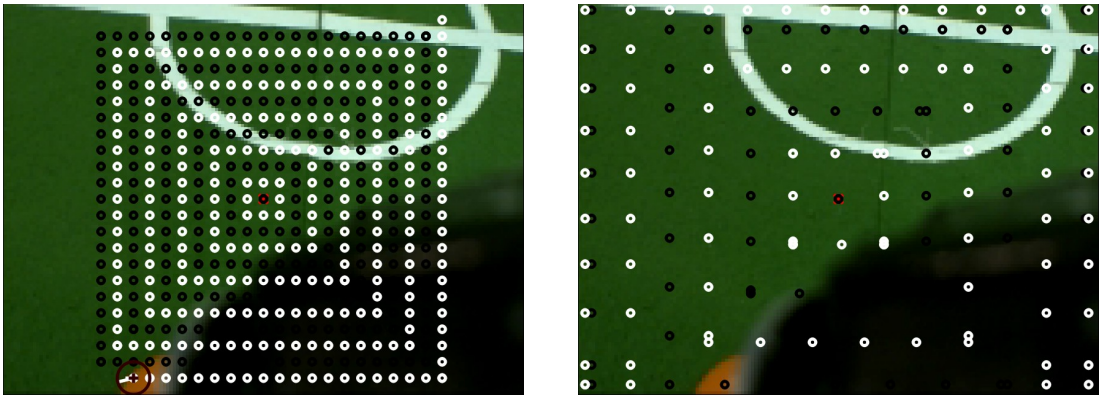
Abbildung 6.8: Detektion des Balls. (a) zeigt die analysierten Punkte im Bild und die Berechnung der Ballparameter aus fünf echten Randpunkten bei Verwendung eines fixen Abstandes zwischen den Suchpunkten von 5 Pixel. (b) zeigt gleiches für den Ansatz mit variabler Punkteanordnung. Zu erkennen ist, dass hierbei für ein äquivalentes Ergebnis wesentlich weniger Pixel klassifiziert werden müssen.

nochmals mit dem tatsächlichen Ballradius verglichen und der Ball als nicht erkannt gewertet, wenn eine zu große Differenz besteht. Stimmen die beiden Radien einigermaßen überein, wird ein Ball-Perzept generiert und den weiteren Komponenten der *RoboCup*-Anwendung zur Verfügung gestellt.

Der `ActiveBallPerceptorVarGrid` legt im Wesentlichen die gleiche Strategie zugrunde, nur dass bei der Suche nach dem Ball die Variante mit dem flexiblen Raster verwendet wird, wie sie im Abschnitt 6.4.1 über den Basis-Objekt-Perzeptor beschrieben wird. Das hat auch zur Folge, dass die Suche wirklich erst dann beendet wird, wenn ein Ball (und nicht nur eine Ansammlung orangener Pixel) detektiert wurde. Abbildung 6.8 und 6.9 zeigen Beispiele der Balldetektion mit den unterschiedlichen Perzeptoren. Man sieht, dass beide Realisierungen Vor- und Nachteile aufweisen.

6.4.4 Feldlinien-Perzeptor

Das Generieren von Perzepten durch den Feldlinien-Erkenner läuft in der aktuellen Implementierung grundlegend anders ab, als bei den beschriebenen Ball-Perzeptoren. Es wird kein Startpunkt für eine Suche anhand von Modelldaten initialisiert, sondern selbstständig gezielt nach diversen Ausgangspunkten gesucht, von denen aus Feldlinien innerhalb des Bildes rekonstruiert werden können. Hier kommen also keine Modelldaten ins Spiel, sondern vielmehr Expertenwissen, welches besagt, dass Feldlinien



(a) (b)

Abbildung 6.9: Detektion des Balls. (a) zeigt die analysierten Punkte im Bild und die berechnete Lage des Balls bei Verwendung eines fixen Abstandes zwischen den Suchpunkten von 5 Pixel. (b) zeigt gleiches für den Ansatz mit variabler Punkteanordnung, allerdings kann hierbei der Ball nicht erkannt werden, da er von keinem Suchpunkt getroffen wird – zumindest wird kein als Orange klassifiziertes Pixel analysiert.

innerhalb des Bildes auch am Bildrand zu finden sind.

Deshalb wird in einem ersten Schritt, der Bildrand nach *Runs* in der Farbe der Feldlinien (Weiß) durchsucht (siehe Abschnitt 6.4.2). Ausgehend von diesen werden sogenannte Kurven (`Curve`) generiert, indem die Ausgangs-*Runs* quasi entlang der Feldlinien expandiert werden (siehe Abschnitt 6.4.2). Im Prinzip geht dieses Expandieren so vonstatten, dass der aktuelle *Run* orthogonal verschoben und dann an dieser Stelle wieder an die sich dort befindende(n) Linie(n) angepasst wird. Die Mehrzahl ist an dieser Stelle möglich, da sich Linien an Kreuzungen auch aufteilen können.

Schon während dieses Prozederes werden aus abgeschlossenen Kurven gerade Linien extrahiert, indem für Punktepaare der Kurven (angefangen mit den Enden) überprüft wird, ob zwischen diesen eine gerade Verbindung besteht. Die Überprüfung findet so statt, dass auf der fiktiven geraden Linie, definiert durch das Punktepaar, eine bestimmte Anzahl weißer Pixel an bestimmten Stellen (zum Beispiel in der Mitte) vorhanden sein muss. Außerdem wird darauf geachtet, dass in diesem Schritt Liniensegmente generiert werden, die eine bestimmte Länge nicht überschreiten, damit ausführlichere Informationen für die spätere Bestimmung des Mittelkreises zur Verfügung stehen. Besteht keine oder eine zu lange Verbindung wird die Kurve sukzessive halbiert und wieder auf eine Verbindung getestet.

Nachdem also möglichst das gesamte Feldliniennetzwerk innerhalb des Bildes durch mehr oder weniger viele Liniensegmente beschrieben wird, muss die wahre

Struktur und Anordnung ermittelt werden. Als erstes wird das Vorkommen des Mittelkreises bestimmt. Dazu wird für alle Paare von Liniensegmenten festgestellt, ob diese auf einem gemeinsamen Kreis liegen können, der in etwa die Ausmaße des Mittelkreises hat. So kommen Kandidaten für den Mittelkreis zustande, deren Parameter gemittelt werden, insofern eine gewisse Konsistenz unter den Kandidaten erreicht wird. Ansonsten wird kein Mittelkreis aufgrund zu hoher Unsicherheiten erkannt.

Anschließend werden die Liniensegmente zu kompletten Linien zusammengefasst, wobei auch Verdeckungen überwunden werden können, indem man davon ausgehen kann, das Liniensegmente, deren Enden alle auf einer Geraden liegen, zu ein und derselben Feldlinie gehören. Zum Schluss werden noch Kreuzungen aus den Feldlinien generiert, wodurch dann alle Perzepte erzeugt wurden, für die der `ActiveLinesPerceptor` zuständig ist:

- Perzepte für Feldlinien
- Perzepte für Kreuzungen (also Schnittpunkte von Feldlinien)
- Perzept für den Mittelkreis

Ein beispielhafter Verlauf bei der Rekonstruktion von Feldlinien innerhalb einer Aufnahme zeigen folgende Abbildungen. Abbildung 6.7 beschreibt, wie die erste Kurve des Bildes generiert wird. In Abbildung 6.10 sind alle erzeugten *Runs* dargestellt. Außerdem sind die Linienfragmente eingezeichnet, die entweder zu einer Kreuzung oder zum Mittelkreis gehören. Letztendlich wird in Abbildung 6.11 die wahrgenommene Struktur der Feldlinien visualisiert.

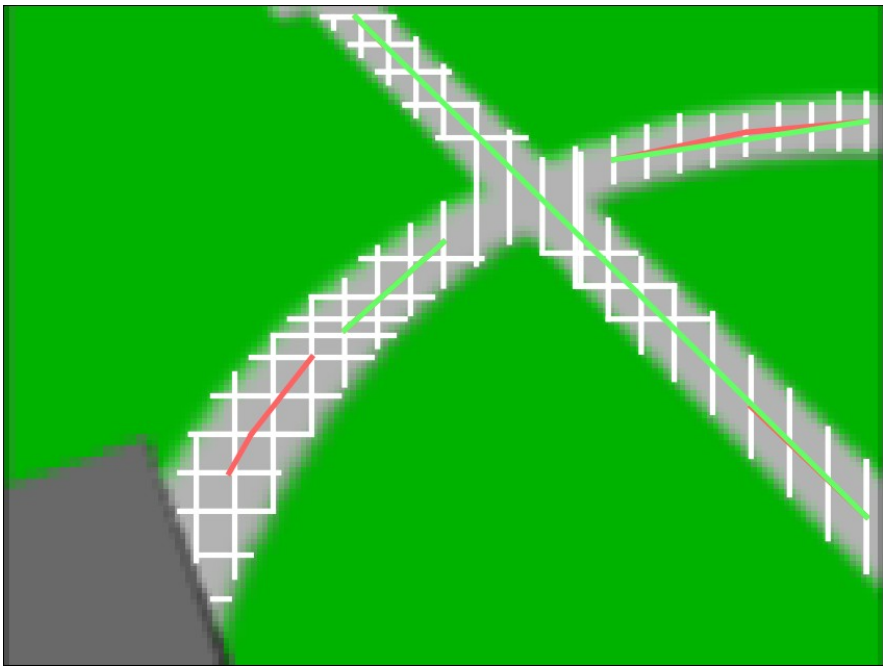


Abbildung 6.10: Vom Feldlinien-Perzeptor generierte Linien-*Runs* und erzeugte Linienfragmente, die entweder zu einer Kreuzung oder zum Mittelkreis gehören.

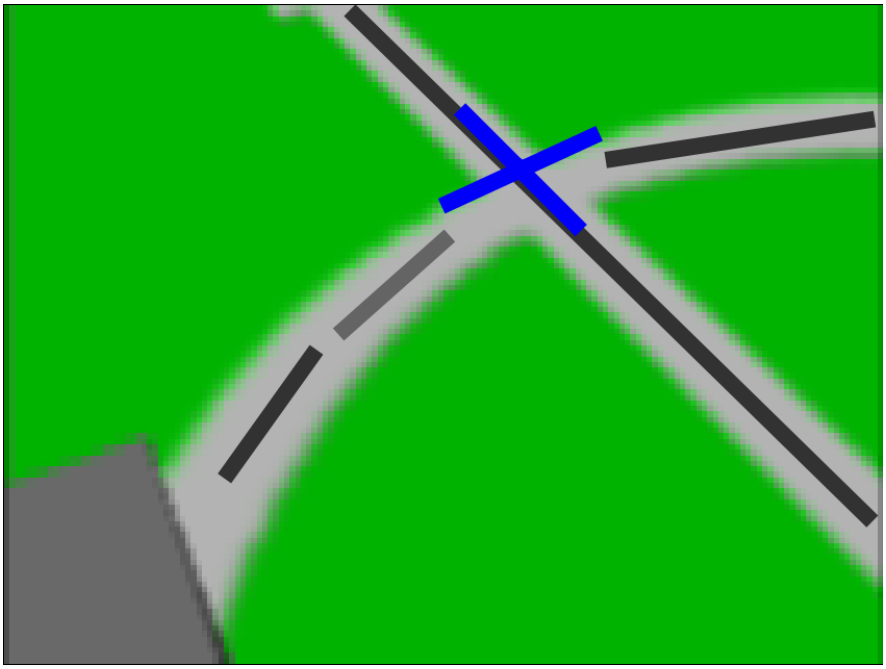


Abbildung 6.11: Beispiel erkannter Feldlinien und detektierter Kreuzung im Bild.

7 Ergebnisse

Die in dieser Arbeit präsentierten Ergebnisse beruhen hauptsächlich auf Tests, die mit Hilfe eines speziell für *RoboCup*-Anwendungen entwickelten Simulators [7] und Zustandsaufnahmen vom realen Roboter durchgeführt worden sind. Die aufgenommenen Zustände beinhalten Kamerabilder und die Kameramatrix, so dass alle für die Bildverarbeitung im Kern relevanten Daten vorhanden sind. Für den Ansatz der ActiveVision zusätzlich benötigte Informationen der Modellierung werden durch die Applikation selbst aufgrund der Daten generiert, die wiederum aus der Bildverarbeitung kommen. Dieses entspricht dem Vorgehen, wie es auch beim Einsatz auf dem realen Roboter der Fall ist.

Das Augenmerk der Tests liegt auf zwei unterschiedlichen Kriterien, für die prinzipiell ein Kompromiss gefunden werden muss, da sie sich in der Regel gegenseitig komplementär bezüglich ihrer Güte verhalten. Das heißt, für eine gesteigerte Erfüllung des einen Zieles müssen Einschränkungen in Bezug auf das andere in Kauf genommen werden. Die beiden zu untersuchenden Ziele dieser Arbeit sind zum einen die Zuverlässigkeit und Robustheit des neuen Ansatzes und zum anderen die Effizienz der Ausführung. Hierfür wird zum Vergleich das bisherige Konzept der Bildverarbeitung (siehe Abschnitt 3.4) herangezogen, wie es vom Team der Darmstadt Dribblers schon sehr erfolgreich im *RoboCup* eingesetzt worden ist. Es steht somit ein tatsächlich einsatz- und vor allem konkurrenzfähiges System zur Verfügung, anhand dessen eine aussagekräftige Bewertung der ActiveVision stattfinden kann.

7.1 Zuverlässigkeit und Robustheit

In diesem Abschnitt soll dargestellt werden, wie zuverlässig Objekte oder Feldlinien innerhalb der Bilder auch bei Bildstörungen oder Verdeckungen erkannt werden. Hierzu werden ausschließlich einzelne Bilder betrachtet, auf die der neue und der alte Ansatz der Bildverarbeitung angewendet werden. Dabei handelt es sich um reale Aufnahmen. Die tatsächliche Projektion der Daten in das Roboterkoordinatensystem zu vergleichen, ist nicht notwendig, da dieses bei beiden Ansätzen nach dem exakt gleichen Prinzip geschieht. Es reicht also eine Gegenüberstellung der extrahierten Informationen bezüglich der aufgenommen Bilder.

7.1.1 Zuverlässigkeit und Robustheit des Ball-Perzeptors (mit flexiblem Suchraster)

Da Der Ball-Perzeptor, dessen Suche auf einem flexiblem Raster basiert, quasi die Weiterentwicklung des Erkenners mit fixem Raster ist, beziehen sich die Tests in diesem Abschnitt ausschließlich auf diesen. Ziel der ActiveVision ist in erste Linie eine Verbesserung der Performanz. Deshalb soll an dieser Stelle lediglich gezeigt werden, dass die Erkennung des Balls ungefähr gleichwertig zum vorherigen Ansatz ist.

Im Folgenden werden verschiedene Beispiele aufgezeigt werden, die das Verhalten der beiden unterschiedlichen Konzepte repräsentativ gegenüberstellen. Es ist zu erkennen, dass je nach Situation der Ball mal gleich gut detektiert wird und mal sich der eine, mal sich der andere Ansatz als vorteilhaft erweist. Ein erkannter Ball wird jeweils durch einen eingezeichneten Kreis dargestellt. Konnte kein Ball erkannt werden, werden die analysierten Pixel visualisiert.

Die ersten Beispiele zeigen, dass die Genauigkeit der Ballerkennung in unterschiedlichen Situationen gleichwertig ist. Ist der Ball komplett zu sehen und beinhaltet kaum Bildfehler, ist die Erkennung für beide Ansätze recht unproblematisch (siehe Abbildung 7.1). Auch wenn nur Teile des Balls zu sehen sind, werden in der Regel äquivalente Ergebnisse erzielt (siehe Abbildung 7.2). Abbildung 7.3 stellt abschließend noch einmal dar, dass auch bei groben Bildstörungen im Bereich des Balls beide Konzepte in der Lage sein können, die Parameter richtig zu bestimmen.

Es gibt allerdings auch Situationen in denen der bisherige Ansatz bessere Ergebnisse liefert, was einfach darauf zurückzuführen ist, dass hier mehr Informationen aus dem Bild extrahiert werden. Durch die derzeitige Implementierung der Klasse `ActiveBallPerceptorVarGrid` kann es passieren, das ein verdeckter Ball übersehen wird, da ein relativ großer Abstand zwischen den Suchpunkten gewählt wurde (siehe Abbildung 7.4). In Abschnitt 8.2 wird aber eine Möglichkeit vorgestellt, wie man dem entgegenwirken könnte.

Die restlichen Beispiele zeigen Situationen, in denen die ActiveVision dem bisherigen Ansatz überlegen ist. Zum einen kann dieses in Fällen auftreten, in denen der Ball ungünstig verdeckt oder verrauscht im Bild erscheint (siehe Abbildung 7.5) und zum anderen, wenn der Ball weit entfernt ist und nur noch als wenige Pixel im Bild auftritt (siehe Abbildung 7.6). Diese positive Eigenschaft des neuen Konzepts kann damit erklärt werden, dass es sich implizit auf Regionen besonderen Interesses (zu englisch: „regions of interest" oder ROIs) konzentriert.

Als Fazit lässt sich ziehen, dass beide Ansätze bezüglich der Ballerkennung Vor- und Nachteile aufweisen. Keiner der beiden kann immer perfekte Ergebnisse liefern, aber prinzipiell sind beide als gleich zuverlässig zu bewerten. Ein Vorteil des bisherigen Konzepts ist, dass in der Regel viel mehr Informationen aus dem Bild extra-

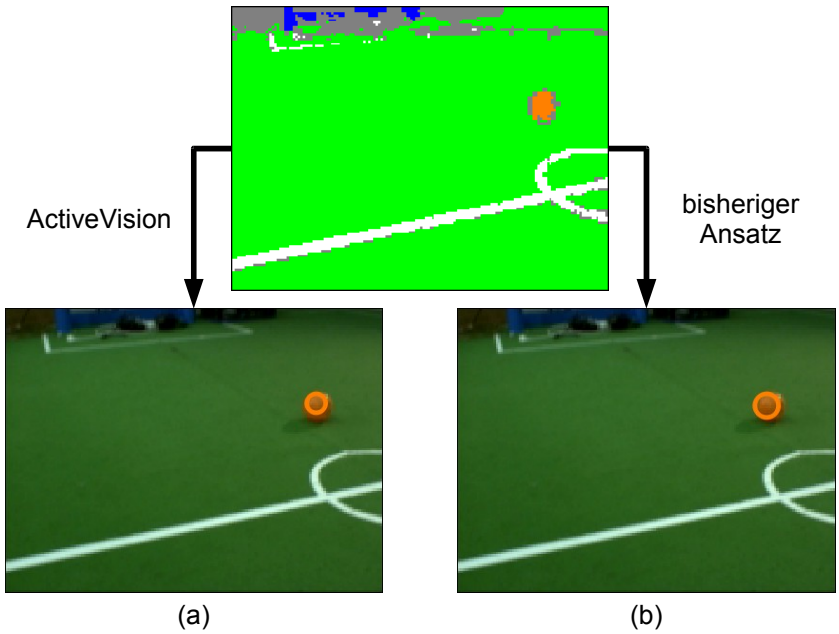


Abbildung 7.1: Beispiel zur Ballerkennung mit dem Ansatz der ActiveVision (a) und dem bisherigen (b). Hier ist der Ball komplett und ohne größere Störungen zu sehen und wird dementsprechend gut von beiden Konzepten erkannt.

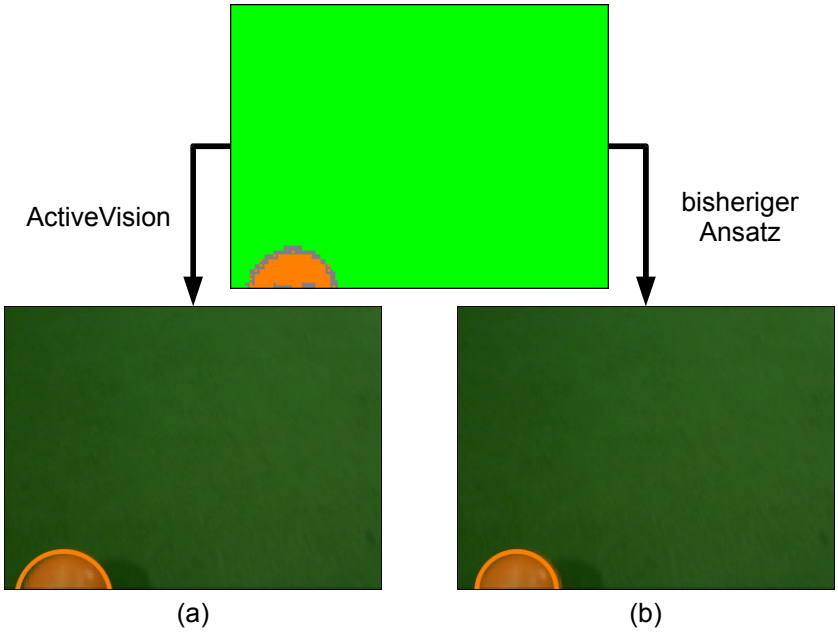


Abbildung 7.2: Beispiel zur Ballerkennung mit dem Ansatz der ActiveVision (a) und dem bisherigen (b). Hier ist der Ball zwar nur Teilweise zu sehen, kann aber von beiden Konzepten gut erkannt werden.

hiert werden und dieses immer auf die exakt gleiche Weise passiert. Dadurch wird die Ballerkennung vorhersagbarer und es besteht weniger die Gefahr, dass ein ruhender Ball an leicht unterschiedlichen Stellen detektiert wird. Das kann beim Ansatz der ActiveVision derzeit eher vorkommen, da hier die Rekonstruktion des Balls von dem Punkt abhängt, der als erstes dem Ball zugeordnet werden konnte. Im Ausblick wird eine mögliche Lösung dieses Problems vorgestellt werden (siehe Abschnitt 8.2).

Nun aber zu den Vorteilen des ActiveVision-Ansatzes gegenüber dem bisherigen Konzept. Der größte Gewinn liegt wie gefordert darin, dass in der Praxis wesentlich weniger Pixel des Bildes analysiert werden müssen, um den Ball zu finden. Aber auf diesen Aspekt wird in Abschnitt 7.2 genauer eingegangen. Es gibt jedoch auch Vorteile bezüglich der Erkennungsgenauigkeit. Dadurch dass nicht erst ein starres Raster über das Bild gelegt wird und danach die Erkenner auf den schon extrahierten Daten arbeiten müssen, kann an bestimmten Stellen gezielt eine genauere Untersuchung des Bildes vorgenommen werden. So ist es zum Beispiel möglich einen entfernten Ball besser zu erkennen, da schon beim (vorläufigen) Auffinden eines einzelnen orangenen Pixels der übergeordnete Suchvorgang angehalten wird. Es wird dann versucht eine zuverlässige Rekonstruktion des Balls zu finden, und nur wenn das misslingt, wird die Suche fortgesetzt. Im bisherigen Ansatz fällt ein Ball, der von zu wenigen *Scan-Lines* geschnitten wird, im wahrsten Sinne des Wortes durchs Raster. Dieser Aspekt führt auch teilweise zu einer genaueren Bestimmung der Ballparameter, wenn eine Verde-

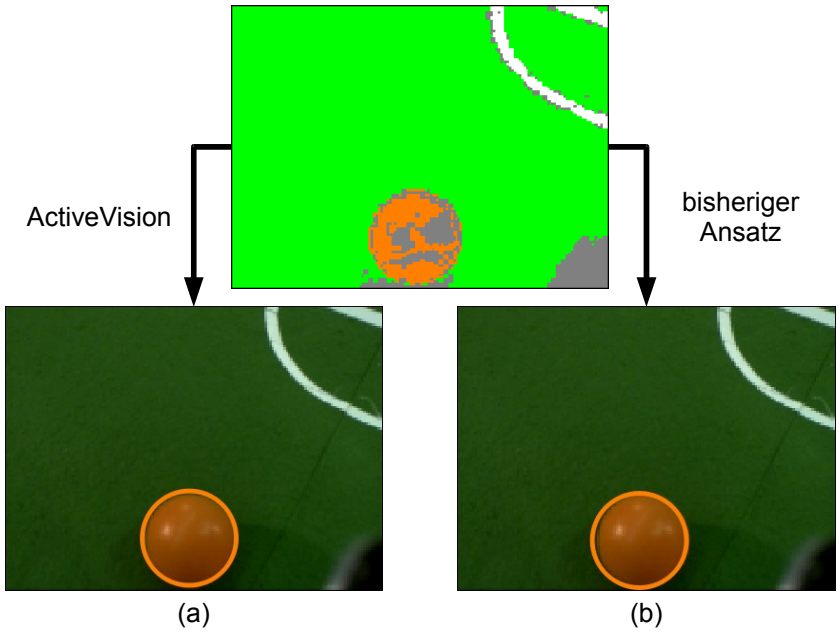


Abbildung 7.3: Beispiel zur Ballerkennung mit dem Ansatz der ActiveVision (a) und dem bisherigen (b). Trotz stärkerer Beeinträchtigungen der Ballzugehörigen Pixel, wird er von beiden Konzepten zuverlässig erkannt.

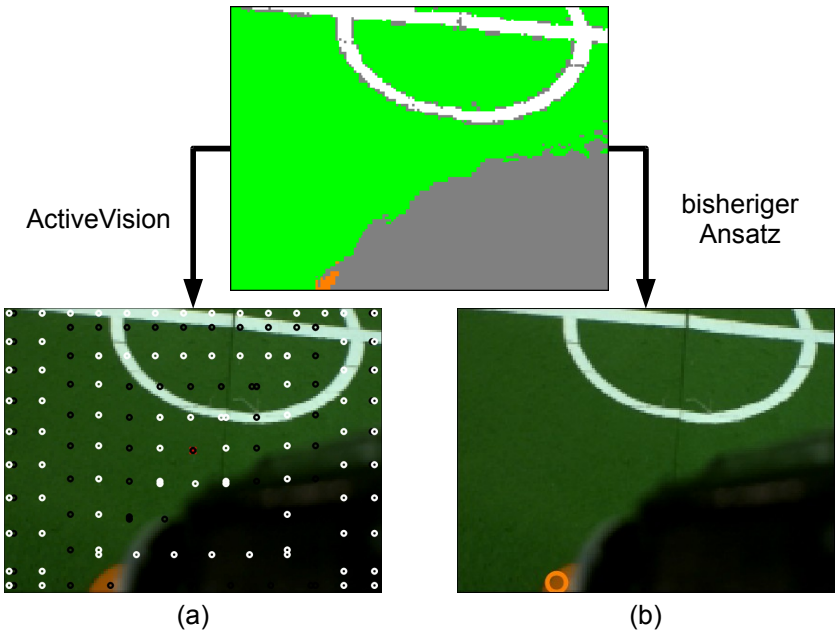


Abbildung 7.4: Beispiel zur Ballerkennung mit dem Ansatz der ActiveVision (a) und dem bisherigen (b). Hier wird der stark verdeckte Ball lediglich vom bisherigen Konzept erkannt. Für die ActiveVision sind deshalb die analysierten Pixel visualisiert worden.

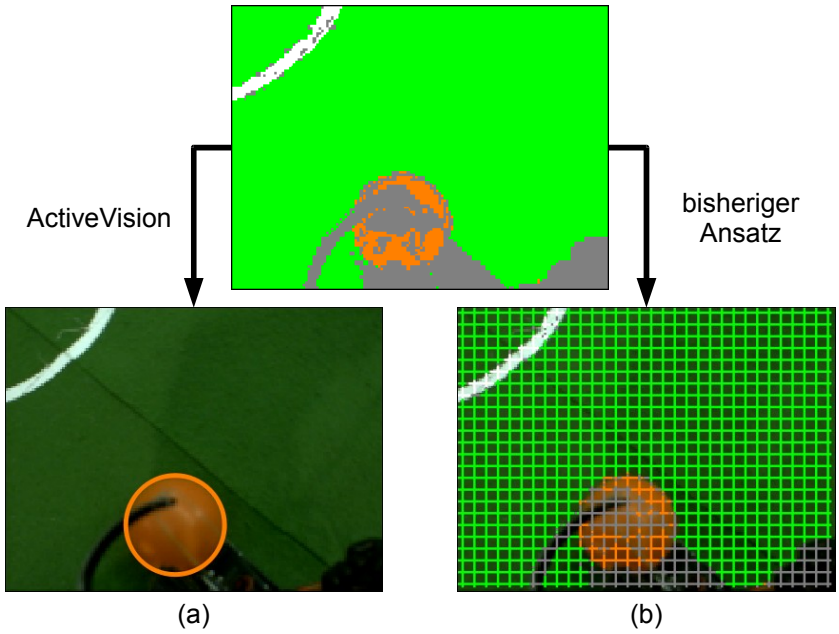


Abbildung 7.5: Beispiel zur Ballerkennung mit dem Ansatz der ActiveVision (a) und dem bisherigen (b). Hier wird der stark verrauschte und leicht verdeckte Ball ausschließlich von der ActiveVision erkannt. Für das bisherige Konzept ist deshalb das Gitter der *Scan-Lines* visualisiert worden.

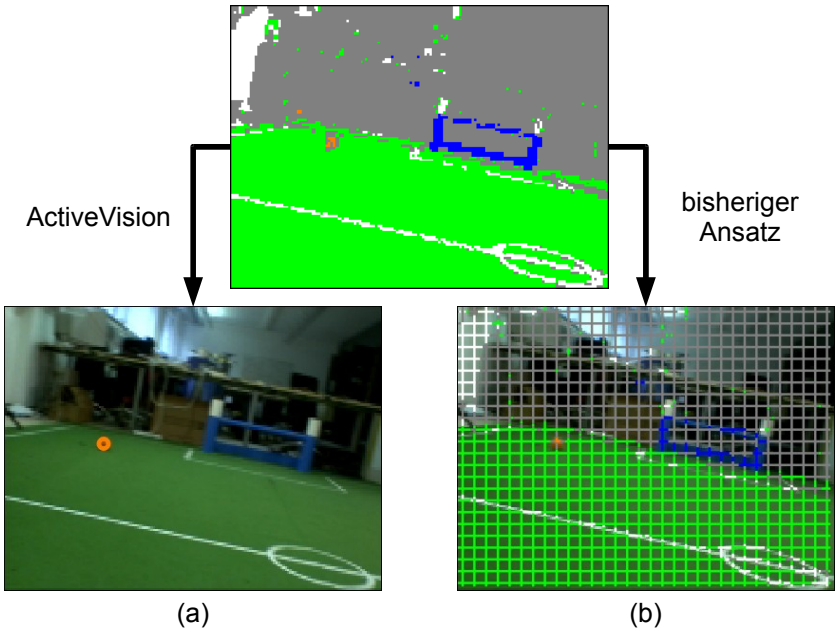


Abbildung 7.6: Beispiel zur Ballerkennung mit dem Ansatz der ActiveVision (a) und dem bisherigen (b). Hier wird der sehr kleine Ball ausschließlich von der ActiveVision erkannt. Für das bisherige Konzept ist deshalb das Gitter der *Scan-Lines* visualisiert worden.

ckung oder starke Bildfehler auftreten, da nicht eine gesamte Punktwolke orangener Pixel verarbeitet wird, sondern gezielt zuverlässige Punkte am Rand des Balls gesucht werden, mit denen sich die Parameter berechnen lassen.

7.1.2 Zuverlässigkeit und Robustheit des Feldlinien-Perzeptors

Vorneweg muss an dieser Stelle darauf hingewiesen werden, dass aus organisatorischen Gründen bei den hier vorgestellten Testfällen ausschließlich Aufnahmen des Spielfeldes der Four-Legged League verwendet werden konnten. Die Entscheidung viel hier auf solche und nicht auf Simulator-generierte, da dadurch Robustheit und Zuverlässigkeit besser überprüft werden können. Dennoch ist nicht auszuschließen, dass auf dem tatsächlichen Humanoiden-Spielfeld geringfügig andere Ergebnisse erzielt werden.

Für die Erkennung von Feldlinien gilt prinzipiell Ähnliches wie schon für die Balldetektion. Beide Ansätze weisen bezüglich der Erkennungszuverlässigkeit und -robustheit Vor- und Nachteile auf. Insgesamt lässt sich aber sagen, das der bisherige Ansatz besser mit fehlklassifizierten Pixeln umgehen kann, die nicht als Weiß (die Farbe der Feldlinien) erkannt werden (siehe Abbildung 7.7). Dieser Umstand lässt sich aber beheben, indem eine bessere Farbtabelle erstellt wird, bei der missklassifizierte Bereiche innerhalb von Linien eher die Ausnahme sein sollten. Dann ist die Linienerkennung als gleichwertig zu betrachten (siehe Abbildung 7.8 und 7.9). Außerdem wird in Abschnitt 8.2 noch auf Verbesserungsmöglichkeiten diesbezüglich eingegangen.

Aber auch ohne weitere Anpassungen gibt es nicht wenige Situation, in denen der `ActiveLinesPerceptor` Feldlinien beziehungsweise deren Zusammenhänge besser erkennt (siehe Abbildung 7.10 und 7.11). Das kann zum einen damit erklärt werden, dass bei der Detektion von Kreuzungen mit den projizierten Linien gerechnet wird und (schon dadurch bedingt) größere Toleranzen zulässig sind, da bekannt ist, dass hier eigentlich nur zwei Fälle vorkommen können. Entweder zwei Linien stehen circa in einem Winkel von 90° oder in 180° respektive 0° zueinander.

Da Linien auch des Öfteren während eines Spiels teilverdeckt sein können, wurden beide Konzepte auf Robustheit gegenüber Verdeckungen verglichen. Abbildung 7.12 zeigt, dass auch der neue Ansatz durchaus das Potenzial hat damit zurechtzukommen. Deshalb sind auch hier keine gravierenden Verschlechterungen zum bisherigen Ansatz zu befürchten.

Der große Vorteil des neuen Feldlinien-Perzeptors gegenüber dem alten liegt darin, den Mittelkreis häufiger detektieren zu können, ohne dabei häufige Fehlerkennungen auszulösen. Hier werden in der Regel bessere Ergebnisse erzielt, wenn nur Teile des

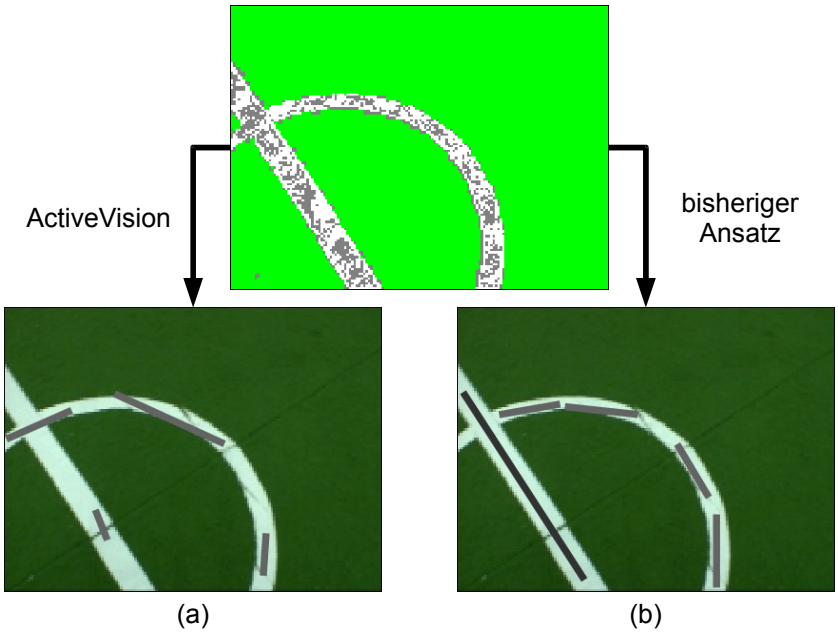


Abbildung 7.7: Beispiel zur Feldlinienerkennung mit dem Ansatz der ActiveVision (a) und dem bisherigen (b). Zu erkennen ist, dass das bisherige Konzept die stark verrauschten Feldlinien besser wahrnimmt.

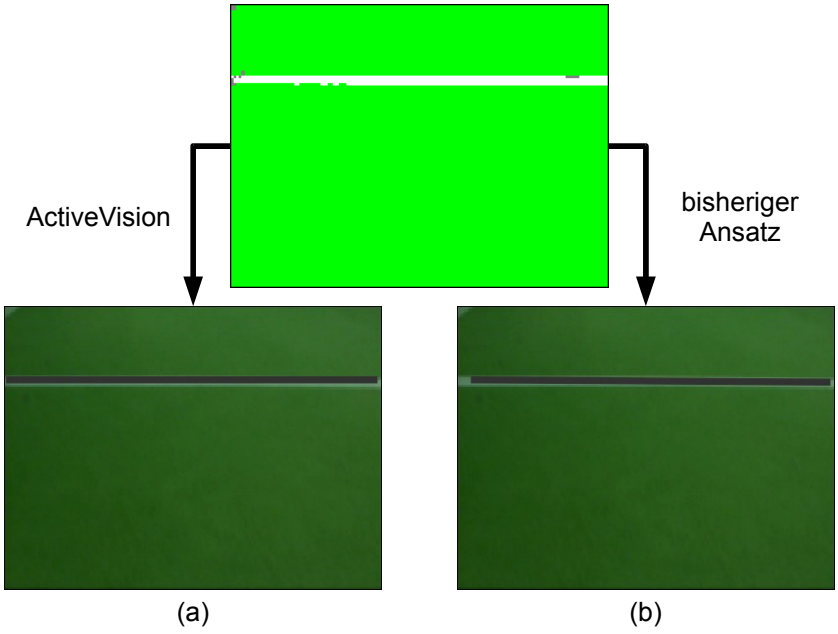


Abbildung 7.8: Beispiel zur Feldlinienerkennung mit dem Ansatz der ActiveVision (a) und dem bisherigen (b). Hier wird eine gut zu erkennende Feldlinie von beiden Konzepten detektiert.

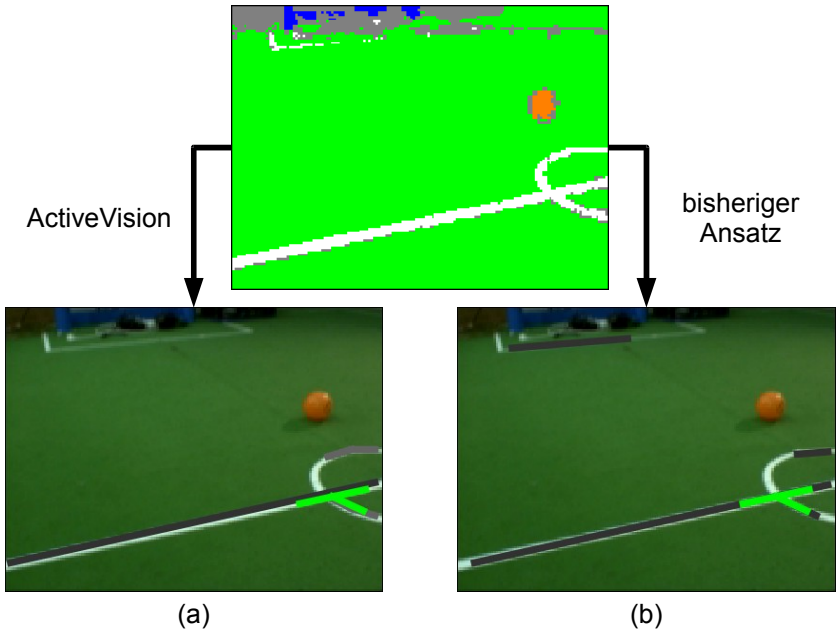


Abbildung 7.9: Beispiel zur Feldlinienerkennung mit dem Ansatz der ActiveVision (a) und dem bisherigen (b). Hier wird die Struktur der Feldlinien im Vordergrund von beiden Konzepten äquivalent erkannt. Allerdings bleibt die vom bisherigen Ansatz detektierte Linie im Hintergrund für die ActiveVision unerreichbar.

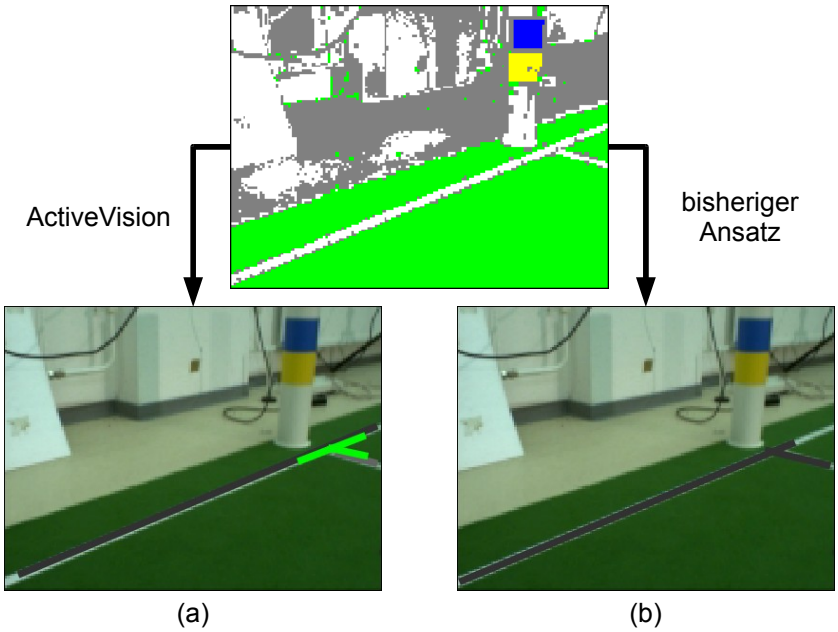


Abbildung 7.10: Beispiel zur Feldlinienerkennung mit dem Ansatz der ActiveVision (a) und dem bisherigen (b). Hier wird die im Bild vorhandene T-Kreuzung ausschließlich von der ActiveVision erkannt.

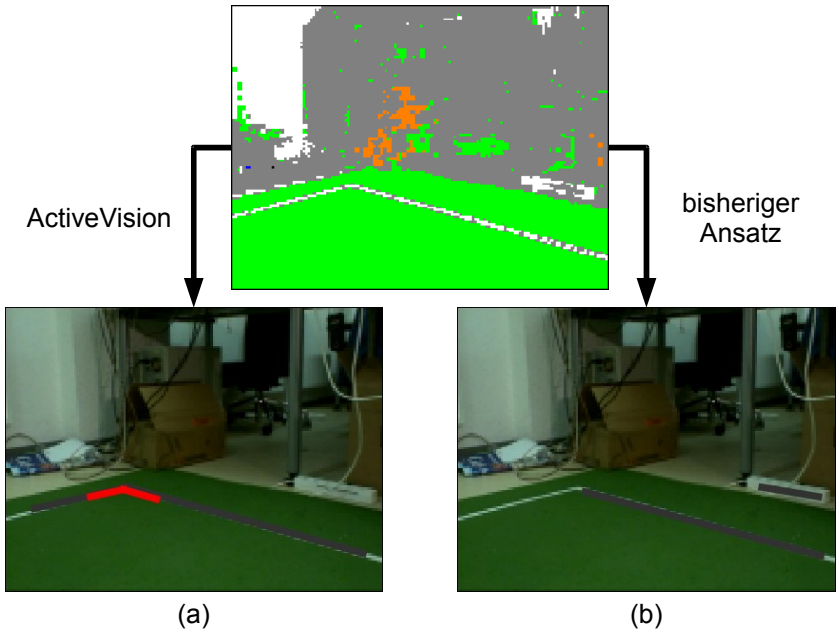


Abbildung 7.11: Beispiel zur Feldlinienerkennung mit dem Ansatz der ActiveVision (a) und dem bisherigen (b). Hier wird eine Feldlinie (und dementsprechend eine Ecke) vom Konzept der ActiveVision erkannt, die vom bisherigen nicht wahrgenommen wird.

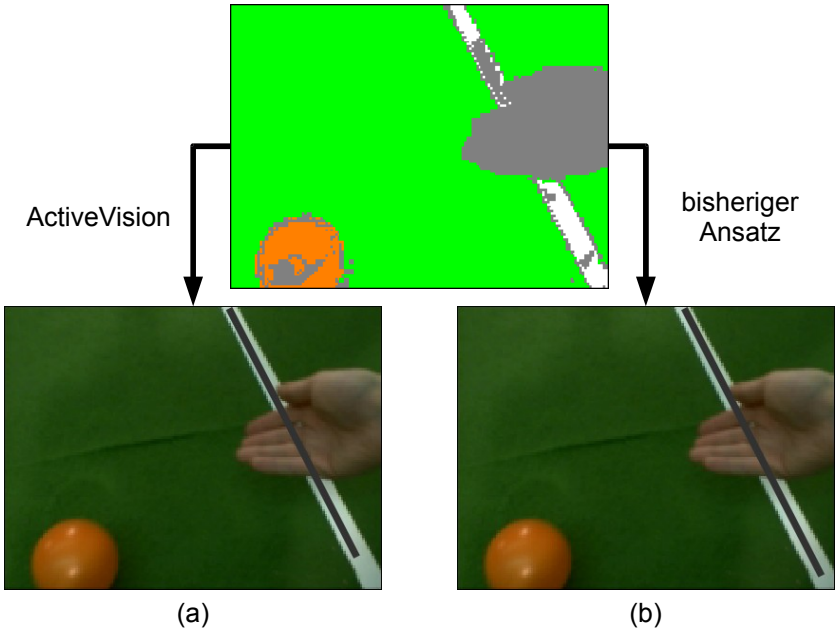


Abbildung 7.12: Beispiel zur Feldlinienerkennung mit dem Ansatz der ActiveVision (a) und dem bisherigen (b). Beide Ansätze können die starke Verdeckung der Linie kompensieren.

Kreises zu sehen sind oder er weiter entfernt ist (siehe Abbildung 7.13[1] und 7.14). Ist der Mittelkreis hingegen allgemein gut zu erkennen, sind beide Ansätze gleichwertig (siehe Abbildung 7.15).

7.2 Effizienz

Für die Effizienzmessungen wurde zum einen das Bildverarbeitungsmodul der ActiveVision und zum anderen das bisherig, der `GridImageProcessor`, in Schleifen von 200 Durchläufen pro Bild ausgeführt, damit die Messungen aufgrund sehr kleiner Zeiten nicht verfälscht wurden. Zur Messung selbst wurden eigene Marken vor und hinter die eigens eingefügten Schleifen im Quellcode gesetzt, zwischen denen dann anhand der vom *RoboFrame* [13] zur Verfügung gestellten Werkzeuge der zeitliche Abstand bei der Ausführung bestimmt wurde. Zwecks besserer Vergleichbarkeit befinden sich die Marken (und somit die Schleife) erst nach den Tests, die bei beiden Modulen am Anfang der Ausführung durchlaufen werden und entscheiden, ob es überhaupt zur eigentlichen Bildverarbeitung im aktuellen Zyklus kommt.

[1] Als kurze Anmerkung sei noch erwähnt, dass die in den Abbildungen dargestellten Radien nicht stimmen, da wie gesagt auf das Feld der Four-Legged League zurückgegriffen werden musste.

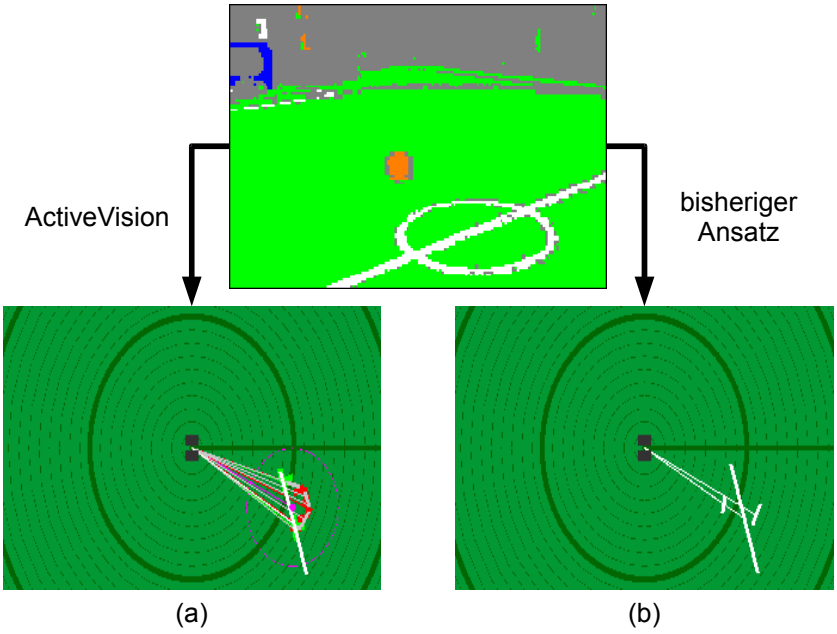


Abbildung 7.13: Beispiel zur Detektion des Mittelkreises mit dem Ansatz der ActiveVision (a) und dem bisherigen (b). Hier wird der vorhandene Mittelkreis ausschließlich von der ActiveVision erkannt.

Die Messungen wurden anhand zweier Testfälle im Simulator vorgenommen. In beiden Fällen wird das Verhalten ausgeführt, welches in Spielen des *RoboCup* für den Stürmer eingesetzt wird. Hierbei sucht der Roboter den Ball, geht zu ihm und schießt ihn in Richtung des gegnerischen Tors. Es wurden allerdings ausschließlich ein Ball- und ein Feldlinien-Perzeptor von den Bildverarbeitungsmodulen ausgeführt. Der Startpunkt des Roboters war immer gleich, nämlich genau im Zentrum des Mittelkreises. Der Unterschied der beiden Testszenarien lag darin, dass der Ball einmal 50 cm hinter dem Roboter platziert wurde und einmal gar nicht vorhanden war. Der erste Fall führte dazu, dass der Ball in vielen – vor allem aufeinander folgenden – Bildern zu erkennen war. Da dieses besonders dem eingesetzten `ActiveBallPerceptorVarGrid` zugute kommt (siehe Abbildung 7.16), aber nicht in jeder Situation eines realen Spiels zutrifft, wurde das zweite Szenario gewählt, in dem nie ein Ball erkannt werden konnte.

Die Ergebnisse der Messungen haben jedoch gezeigt, dass die Ausführungszeiten des bisherigen Ansatzes in beiden Testszenarien vom ActiveVision-Ansatz deutlich unterboten werden. Dieses ist eine direkte Auswirkung des Konzepts, durch das in der Praxis häufig signifikant weniger Pixel im Bild analysiert werden müssen. Aber selbst die Suche nach dem Ball im zweiten Testfall, die jedes Mal das ganze Bild durchläuft, da kein Ball vorhanden war, verlangsamt die gesamte Bildverarbeitung der ActiveVision nicht so sehr, dass die Ausführungszeiten mit denen des vorherigen An-

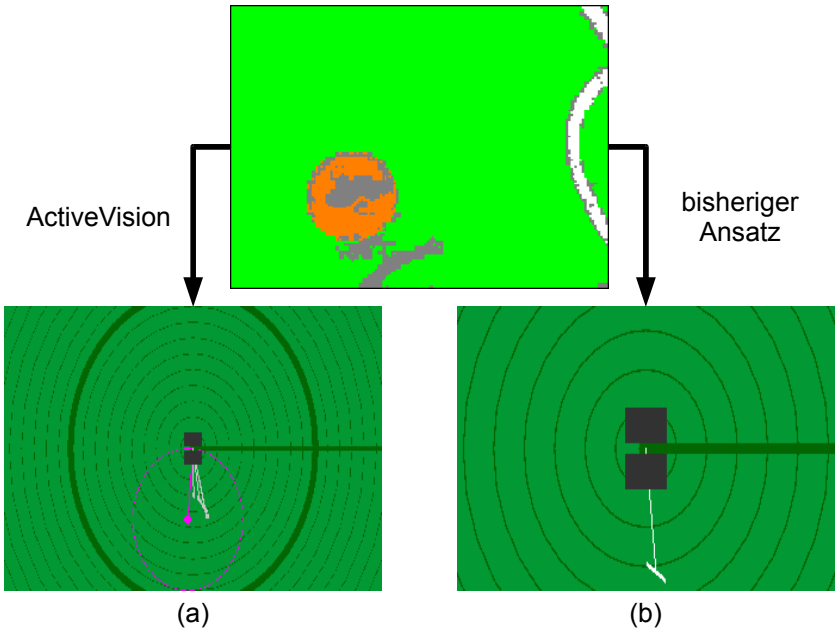


Abbildung 7.14: Beispiel zur Detektion des Mittelkreises mit dem Ansatz der ActiveVision (a) und dem bisherigen (b). Hier wird der nur teilweise vorhandene Mittelkreis ausschließlich von der ActiveVision erkannt.

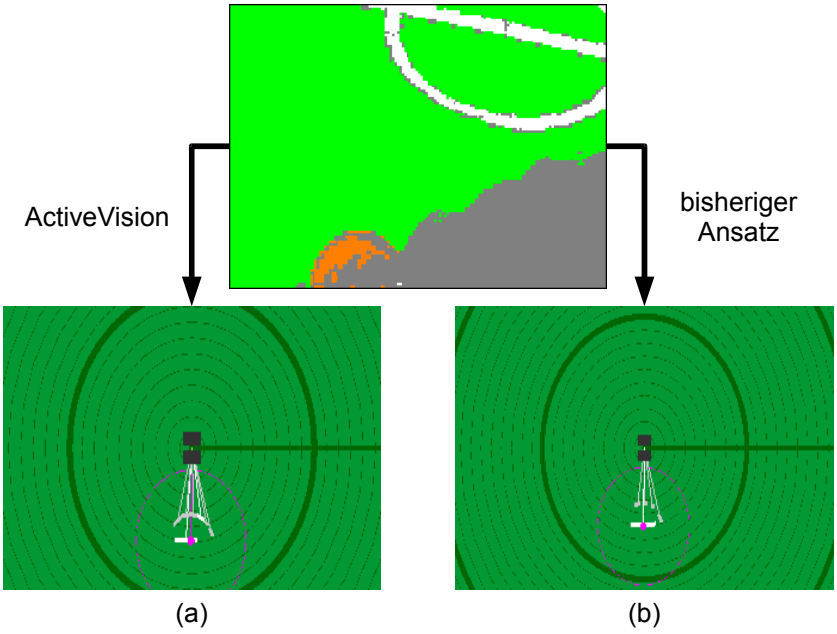


Abbildung 7.15: Beispiel zur Detektion des Mittelkreises mit dem Ansatz der ActiveVision (a) und dem bisherigen (b). Hier wird der vorhandene Mittelkreis durch beide Konzepte korrekt erkannt.

satzes zu vergleichen wären.

Tabelle 7.1 und 7.2 fassen die gemessenen Zeiten noch einmal in Zahlen zusammen. Es wird darauf verzichtet, einzelne Ergebnisse anzugeben. Die Messungen wurden auf einem Notebook (Dual Core 1,6 GHz, 1 GB RAM) durchgeführt und ergeben, dass die Ausführung des ActiveVision-Moduls im Durchschnitt bis zu 26-mal schneller ist als die des `GridImageProcessor`'s. Aber selbst wenn man innerhalb der Messreihen die längste Ausführungsdauer des `ActiveImageProcessor`'s mit der kürzesten des alten Ansatzes vergleicht, besteht eine prozentuale Differenz von über 140% zugunsten des neuen Konzepts.

Es fällt auf, dass sich beide Szenarien unterschiedlich auf die verschiedenen Konzepte auswirken. Dieses war aber zu erwarten, da der bisherige Ansatz unabhängig von der Situation immer das komplette Bildraster analysiert, und sich erst danach im Szenario ohne Ball die Berechnung der entsprechenden Parameter sparen kann. Der ActiveVision-Ansatz hingegen profitiert davon, dass Modelldaten über das gesuchte Objekt vorhanden sind, wodurch es schneller detektiert werden kann. Außerdem wird die Suche beendet, sobald das Objekt identifiziert wurde. Aus diesen Gründen verkürzen sich dessen Ausführungszeiten im ersten Testfall und bewirken einen noch größeren Gewinn gegenüber dem aktuellen Ansatz.

Eine weitere Testreihe wird in Tabelle 7.3 beschrieben. Hierbei lagen keine Simula-

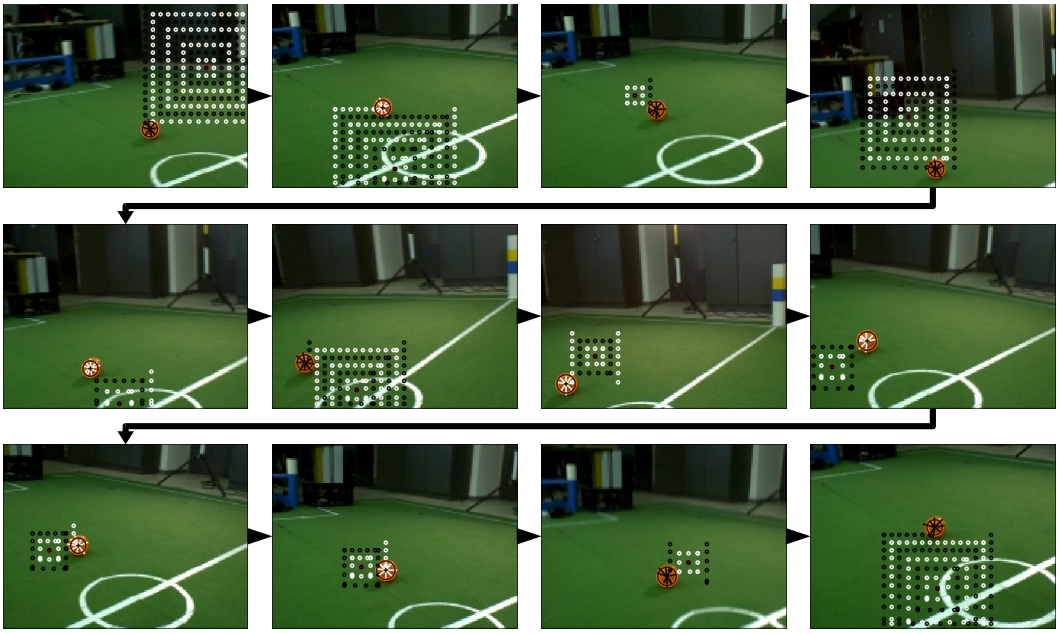


Abbildung 7.16: Sequenz von Aufnahmen, in denen der Ball detektiert worden ist, während der Roboter auf ihn zuläuft. Hier werden die bei der Suche analysierten Pixel visualisiert. Ausgangspunkt war jeweils die aktuell geschätzte Position des Balls (die aus den vorherigen Aufnahmen resultiert). Es ist gut zu erkennen, dass immer nur ein Teil des Bildes verarbeitet werden muss. Selbst in den Fällen, in denen die Schätzung nicht sehr genau ist (des Öfteren durch die Laufbewegung des Roboters bedingt), besteht noch ein enormer Vorteil gegenüber einer kompletten Analyse des Bildes.

	ActiveVision-Ansatz	bisheriger Ansatz
Durchschnitt	25 ms	650 ms
Minimum	15 ms	390 ms
Maximum	46 ms	890 ms

Tabelle 7.1: Gemessene Ausführungszeiten beim Testszenario mit Ball.

	ActiveVision-Ansatz	bisheriger Ansatz
Durchschnitt	47 ms	342 ms
Minimum	15 ms	187 ms
Maximum	140 ms	765 ms

Tabelle 7.2: Gemessene Ausführungszeiten beim Testszenario ohne Ball.

tordaten zugrunde, sondern die Bildverarbeitung fand auf realen Aufnahmen statt. Ein Ausschnitt dieser Sequenz ist in Abbildung 7.16 zu sehen. Auch bei diesem Testszenario ergibt sich ein erheblicher Performanzvorteil des ActiveVision-Moduls gegenüber dem bisherigen Ansatz. Damit wird gezeigt, dass ebenfalls im realen Einsatz großes Potenzial im ActiveVision-Konzept steckt. Die Ergebnisse aller drei Testfälle sind in Abbildung 7.17 noch einmal zusammengefasst.

	ActiveVision-Ansatz	bisheriger Ansatz
Durchschnitt	32 ms	274 ms
Minimum	15 ms	156 ms
Maximum	46 ms	437 ms

Tabelle 7.3: Gemessene Ausführungszeiten bei der Verarbeitung realer Aufnahmen.

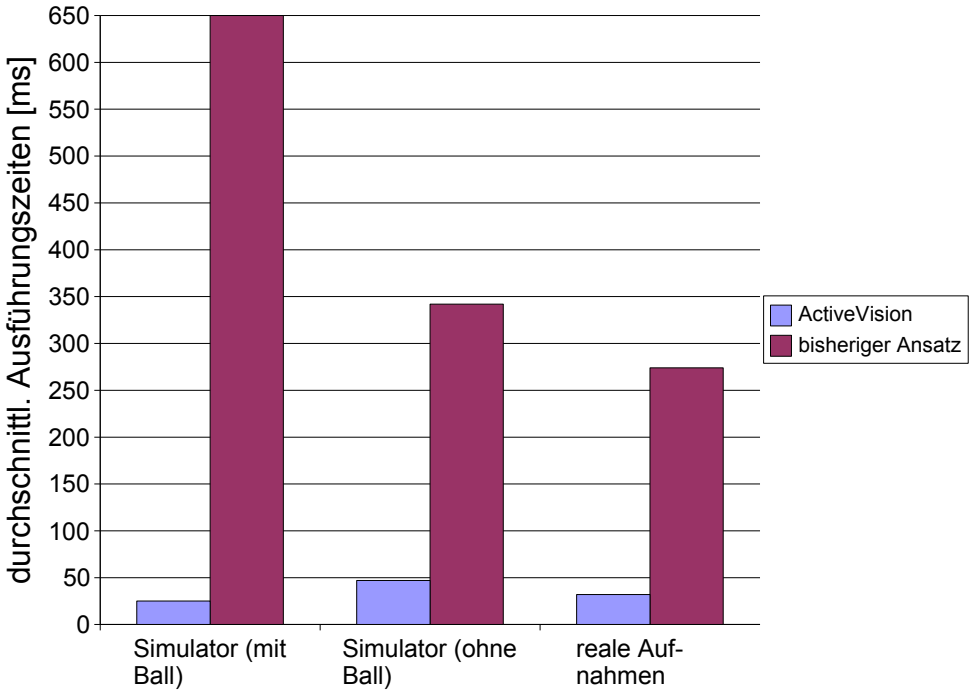


Abbildung 7.17: Vergleich der durchschnittlichen Ausführungszeiten bei den unterschiedlichen Testszenarien.

8 Zusammenfassung und Ausblick

8.1 Zusammenfassung

In Kapitel 3 wurden verschiedene Ansätze gezeigt, wie sie für den *RoboCup* entwickelt und auch schon eingesetzt worden sind. Hierbei gilt es konventionelle Verfahren und neuere innovative Konzepte zu unterscheiden. Oft sind die Grenzen hier aber fließend, da in der Regel nicht jedes Detail neu entworfen wird. So ist es zum Beispiel weiterhin Standard auch bei neueren Ansätzen eine Farbtabelle zu verwenden, um rohe Farbwerte effizient zu klassifizieren, wie es ebenfalls die ActiveVision tut. Vergleicht man das Konzept der ActiveVision weiter mit den vorgestellten Ansätzen so können teilweise weitere Gemeinsamkeiten festgestellt werden. So wurden Arbeiten vorgestellt, deren Ziel es ist, das Bild gezielter nach Informationen zu untersuchen, mehr oder weniger ähnlich dem Ansatz der ActiveVision (siehe Abschnitt 3.3.3 und 3.3.4). Auch Strategien, die ein angepasstes Suchraster zugrunde legen, wurden beschrieben (siehe Abschnitt 3.3.1 und 3.3.2). Das in dieser Arbeit vorgestellte Konzept der ActiveVision führt diese beiden Verfahren zusammen.

Die ActiveVision steckt allerdings noch in den Kinderschuhen. Es konnten lediglich Prototypen entwickelt werden, die noch der Optimierung bedürfen. Dieses sollte jedoch über einen längeren Zeitraum mit ausführlichen Tests geschehen, was im Rahmen dieser Diplomarbeit aus organisatorischen und zeitlichen Gründen nicht im geforderten Maße geschehen konnte. Erste Tests, die aber schon durchgeführt werden konnten, haben das Potenzial dieses Ansatzes aufgezeigt. Die Zuverlässigkeit und Robustheit bezüglich der Erkennung von relevanten Daten im Bild, kann in den meisten Fällen mit dem bisherigen Ansatz mithalten, ihn manches Mal sogar übertreffen. Hier sollte es aber möglich sein, noch höhere Standards zu erreichen, vor allem wenn man sich entschließt etwas der eingesparten Ausführungszeiten wieder in die Erkennung zu investieren. Denn genau dieser gewonnene Performanzvorteil, konnte wie gefordert erreicht werden. Hier muss nun noch der geeignete Kompromiss zwischen diesen prinzipiell komplementären Zielen gefunden werden.

Was auf jeden Fall gezeigt werden konnte, ist, dass es sich durchaus lohnen kann, den Ansatz der ActiveVision weiter zu verfolgen. Es ist zu sehen, dass ähnliche Er-

kennungsraten und ein beträchtlicher Effizienzgewinn erreicht werden können. Wie der nächste Abschnitt noch darstellen wird, ist das allgemeine Konzept des „Aktiven Sehens" noch lange nicht ausgereizt. Aber selbst die bestehende Implementierung bietet dem Nutzer durch den hohen Grad der Parametrisierung viele Möglichkeiten, die konkrete Umsetzung anzupassen und eventuell geeignetere (Standard-)Paramter (für unterschiedliche Szenarien) zu finden. Zum Beispiel könnte einfach die minimale Schrittweite bei der Suche nach dem Ball angepasst werden, so dass auch entfernte Bälle mit höherer Wahrscheinlichkeit erkannt werden.

Gerade im Hinblick auf die zukünftige Entwicklung im *RoboCup*, die zum Beispiel weniger Landmarken vorsieht, kann es ein immer größerer Vorteil werden, nicht in jedem Fall das gesamte Bild zu analysieren, sondern seine Kapazitäten gezielter einzusetzen. Auch dass die Agilität unter den Robotern stetig zunimmt – das heißt, die Geschwindigkeit allgemein und vor allem das Erreichen des Balls oder strategisch wichtiger Punkte im Speziellen, erfordert gerade am Ball schnell handeln zu können. Diesem Ziel kommt die ActiveVision in besonderem Maße nach, da gerade dem Roboter nahe Bälle sehr erfolgreich und effizient verfolgt werden können.

8.2 Ausblick und mögliche Erweiterungen

Wie schon erwähnt bietet der Ansatz des „Aktiven Sehens" im Allgemeinen, aber auch die konkrete Umsetzung der ActiveVision, noch viele Möglichkeiten das Grundkonzept einer besonders intelligenten Bildverarbeitung auszubauen. Dieses betrifft das Vorgehen der einzelnen Erkenner, aber auch das Zusammenspiel unter den Perzeptoren, sowie das Verhalten von anderen Komponenten der Gesamtanwendung. An dieser Stelle sei noch kurz auf einen Umstand hingewiesen, der die Performanz der ActiveVision beeinflusst. Derzeit kommt es noch manchmal vor, dass die aktuellen Gelenkwinkel des Roboters fehlerhaft vorliegen. Dadurch können Positionen und Längenmaße nicht korrekt berechnet werden. Um eine reibungslose Bildverarbeitung durch die ActiveVision zu gewährleisten, gilt es, diese Datenverfälschungen zu beseitigen.

Robustheit und Zuverlässigkeit der einzelnen Erkenner

Dieser Punkt betrifft vor allem den Feldlinien-Perzeptor, der noch arge Probleme damit hat, wenn Pixel von Feldlinien zu unstet klassifiziert werden. Zum einen kann dieser Schwachpunkt durch gute Farbtabellen gemildert werden, zum anderen besteht die Möglichkeit, bei der Detektion Verfahren anzuwenden, die besser mit derart verrauschten Daten umgehen können. Letzten Endes muss durch Praxistests eruiert werden, welches Maß an Rauschunterdrückung notwendig und sinnvoll ist. Der aktuelle Ansatz dagegen ist in erste Linie auf Performanz getrimmt.

Prinzipiell sind aber folgende Möglichkeiten denkbar, um die Erkennung von Feldlinien robuster zu gestalten. Da Linien ohnehin aus einer Menge von Punkten erzeugt werden, könnte man allein anhand dieser Punkte unterschiedliche Geraden generieren, anstatt zu überprüfen, ob an bestimmten Punkten einer Verbindung die entsprechende Farbe im Bild lokalisiert ist. Auch wäre denkbar, nicht einzelne Punkte zu testen, die eine bestimmte Farbe repräsentieren müssen, sondern mehrere Punkte oder – noch besser – Regionen, von denen nur ein Teil passen muss.

Ein weiteres Problem ist der Umstand, das auch ruhende Objekte eher mal in ihrer erkannten Position „springen" können. Das hängt vor allem damit zusammen, dass die Suche nach Objekten eben nicht immer genau gleich abläuft, sondern an unterschiedlichen Ausgangspunkten gestartet wird. Dadurch kann auch die Rekonstruktion des Objektes zu leicht verschiedenen Resultaten führen. Um hier mehr Ruhe reinzubringen, könnte man nur dann eine neue Suche initiieren, wenn die bisherigen Parameter offensichtlich nicht mehr mit dem Bild übereinstimmen.

„Übersehen" verdeckter Objekte

Es kann durchaus vorkommen, dass ein gesuchtes Objekt „übersehen" wird, obwohl es halbwegs ausreichend im Bild vorhanden ist. Dieses ist der Fall, wenn es teilweise verdeckt wird. Dann kann es zu Situationen kommen, in denen zwei benachbarte Suchpunkte gerade nicht mehr das Objekt treffen (siehe Abbildung 7.4). Um diesem Umstand entgegenzuwirken, müsste detektiert werden, wenn ein analysiertes Pixel zu einem Gegenstand gehört, der das gesuchte Objekt verdecken könnte. In diesem Fall könnte dann die Schrittweite verringert oder sogar gezielt nach dem Ende des Hindernisses gesucht werden, um das Objekt wahrzunehmen, dass eventuell nur direkt am Rand der Verdeckung zu sehen ist. Allerdings tritt dieses Problem in der Praxis eher selten auf, da die gewählten Abstände schon einiges geringer gehalten werden, als die Ausmaße des gesuchten Objektes im Bild sind. Somit ist es wahrscheinlich sinnvoller hinzunehmen, ein Objekt mal in einer einzelnen Aufnahme nicht zu erkennen, als ständig die Ränder von Hindernissen abzusuchen.

Informationsaustausch unter den Erkennern

Da es vorkommen kann, dass ein einzelner Erkenner auf der Suche nach einem Objekt, das komplette Bild analysiert (zum Beispiel weil das Objekt gar nicht im Bild vorhanden ist), besteht die Gefahr, dass Regionen des Bildes nicht nur einmal sondern von mehreren Perzeptoren verarbeitet werden. Um diesen Umstand zu verhindern oder zumindest einzuschränken, wäre es möglich, dass Erkenner untereinander Informationen austauschen. So könnte zum Beispiel der Ball-Erkenner Positionen an den Tor-Erkenner weiterreichen, an denen er die spezifische Farbe detektiert hat. Damit könnte

der ActiveVision-Ansatz so angepasst werden, dass zur Objekterkennung nicht mehr ausschließlich die aktuellen Modelldaten als Ausgangspunkte der Suche dienen, sondern auch oder viel mehr die Informationen eines anderen Perzeptors, die direkt aus dem aktuellen Bild selbst stammen.

Denkbar wäre prinzipiell auch der umgekehrte Fall, in dem ein Perzeptor für eine Region meldet, dass diese nicht weiter verarbeitet werden muss. Gründe dafür könnten sein, dass dort eindeutig nur das vom Perzeptor identifizierte Objekt zu finden ist oder dass weder Hinweise auf das eigene noch auf andere Objekte detektiert worden sind. Allerdings birgt diese Strategie große Risiken und sollte nur zum Tragen kommen, wenn nahezu mit 100%-iger Sicherheit davon auszugehen ist, dass in der besagten Region kein Objekt mehr zu finden ist.

Begrenzung der Suche

Ähnlich dem letztgenannten Punkt des vorherigen Abschnitts, ist auch der folgende Erweiterungsvorschlag mit Vorsicht zu betrachten. Ein gewisses Effizienzpotenzial liegt noch darin verborgen, dass momentan immer das komplette Bild durchsucht wird, wenn das gesuchte Objekt nicht vorhanden ist. Gerade bei stationären Objekten wäre es aber denkbar, ganz auf eine Analyse des Bildes in Bezug auf einzelne Objekte zu verzichten, wenn sie mit hoher Sicherheit nicht im Bild vorhanden sind – der Roboter zum Beispiel in die komplett andere Richtung schaut (vergleiche hierzu auch den Ansatz der Universität von Texas in Abschnitt 3.3.3). Bei einer guten Selbstlokalisierung sollten Fehlentscheidungen diesbezüglich minimiert werden können, insbesondere wenn man die Entscheidung davon abhängig macht, wann die letzte erfolgreiche Suche durchgeführt worden ist. Soll heißen, ein Objekt, dass schon lange nicht mehr detektiert worden ist, wird ab einem gewissen Zeitpunkt wieder in jedem Bild gesucht, bis es wieder gefunden wurde. Diesen Ansatz braucht man aber nicht zu verfolgen, falls man eine zuverlässige Selbstlokalisierung einsetzt und dieser nur glaubt, wenn die geschätzte Position zu fast 100% sicher ist.

Ein anderer Ansatz, der die Selbstlokalisierung miteinbezieht, ist der folgende. Würde über die Selbstlokalisierung und die Kameramatrix vor der Ausführung der einzelnen Perzeptoren festgestellt werden, dass Teile des Bildes nicht mehr zum Spielfeld gehören können, könnte dieses entsprechend berücksichtigt werden. Sicherlich könnte nicht jeder Erkenner eine solche Information ausnutzen (beispielsweise die Gegnererkennung), aber zum Beispiel der Ball braucht nicht außerhalb des Feldes erkannt zu werden.

Gezielte Ausrichtung der Kamera

Führt man den Gedanken des „Aktiven Sehens" konsequent weiter, so stellt sich die Frage, warum erst auf der Ebene der Bildverarbeitung gezielt nach Objekten gesucht werden soll. Bestünde nicht viel mehr die Möglichkeit, schon vorher gewisse Vorkehrungen zu treffen, so dass ein Bild besonders aussagekräftige Informationen beinhaltet? So wäre also eine logische Schlussfolgerung, schon bei der Kameraführung zu berücksichtigen, welche Objekte aktuell besonders relevant sind und wo sie am ehesten zu finden sein könnten. Richtet man die Kameras dementsprechend gezielt nach diesen Kriterien aus, besteht sicherlich noch großes Potenzial, um sich noch besser und effizienter auf dem Spielfeld zurechtzufinden. Zu diesem Thema sei auch noch einmal auf Abschnitt 5.1.2 verwiesen.

Kommunikation zwischen den Robotern

Bei einer zuverlässigen Kommunikation unter den Robotern, könnte auch diese genutzt werden, um den gesamten Erkennungsvorgang effizienter zu gestalten. Das System wäre nicht mehr nur auf eigene Daten angewiesen, sondern könnte auch auf Informationen zurückgreifen, die es selbst gar nicht wahrgenommen hat. So kann zum Beispiel für den Ball eventuell schon eine geeignete Ausgangsposition der Suche innerhalb des Bildes gewählt werden, obwohl dieser im System-eigenen Modell noch nicht vorhanden ist. Oder die Kamera wird gezielt auf den Ball gerichtet, weil ein anderes Teammitglied mitgeteilt hat, wo der Ball liegt.

9 Literaturverzeichnis

[1] BACH, J. ; JÜNGEL, M.: Using pattern matching on a flexible, horizon-aligned grid for robotic vision. In: *Concurrency, Specification and Programming*, 2002

[2] BALCH, T. ; BRUCE, J. ; VELOSO, M.: Fast and inexpensive color image segmentation for interactive robots. In: *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2000

[3] BÜKER, U.: *Aktive Szenenauswertung und Objekterkennung*, Universität Paderborn, Diss., 2001

[4] CHINIFOROOSHAN, E. ; CHITSAZ, H. ; HAJIAGHAI, M.T. ; HEYDARNOORI, A. ; JAMZAD, M. ; KAZEMI, M. ; MIRROKNI, V.S. ; SADJAD, B.S.: A Fast Vision System for Middle Size Robots in RoboCup. In: BIRK, A. (Hrsg.) ; CORADESCHI, S. (Hrsg.) ; TADOKORO, S. (Hrsg.): *RoboCup-2001: Robot Soccer World Cup V*. Berlin : Springer Verlag, 2002, S. 159–203

[5] CHITSAZ, H. ; FOROUGHNASSIRAI, A. ; GHORBANI, R. ; JAMZAD, M. ; KAZEMI, M. ; MIRROKNI, V.S. ; SADJAD, B.S.: Basic Requirements for a Teamwork in Middle Size RoboCup / Computer Engineering Department Sharif Universiy of Technology. Teheran, Iran, 2001. – Technical Description Paper

[6] FRIEDMANN, M. ; KIENER, J. ; PETTERS, S. ; THOMAS, D. ; STRYK, O. von: Team Description for Humanoid KidSize League of RoboCup 2007 / Technische Universität Darmstadt. 2007. – Forschungsbericht

[7] FRIEDMANN, M. ; PETERSEN, K. ; STRYK, O. von: Tailored real-time simulation for teams of humanoid robots. In: *RoboCup Symposium 2007*. Atlanta, GA, USA : Springer-Verlag, Juli 2007

[8] HUNDELSHAUSEN, F. von: *Computer Vision for Autonomous Mobile Robots*, Freie Universität Berlin, Diss., 2004

[9] HUNDELSHAUSEN, F. von ; ROJAS, R.: Tracking Regions. In: BONARINI, A. (Hrsg.) ; BROWNING, B. (Hrsg.) ; POLANI, A. (Hrsg.) ; YOSHIDA, K. (Hrsg.): *RoboCup-2003: Robot Soccer World Cup VII*. Berlin : Springer Verlag, 2004, S. 250–261

[10] JÜNGEL, M. ; RÖFER, T.: Vision-based fast and reactive monte-carlo localization. In: *IEEE International Conference on Robotics and Automation*, 2003, S.

856–861

[11] MELLGREN, F. ; STEGBAUER, M.: *Entwicklung von Modulen für die Bildverarbeitung bei mobilen Robotersystemen*. 2005

[12] MERTSCHING, B.: Aktives Sehen – Eine kurze Einführung. In: *Künstliche Intelligenz* (1999), Nr. 1/99, S. 5–6

[13] PETTERS, S. ; THOMAS, D.: *RoboFrame – Softwareframework für mobile autonome Robotersysteme*. 2005

[14] SRIDHARAN, M. ; STONE, P.: Real-Time Vision on a Mobile Robot Platform. In: *IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2005

[15] STONE, P. ; STRONGER, D.: Selective Visual Attention for Object Detection on a Legged Robot. In: LAKEMEYER, G. (Hrsg.) ; SKLAR, E. (Hrsg.) ; SORENTI, D. (Hrsg.) ; TAKAHASHI, T. (Hrsg.): *RoboCup-2006: Robot Soccer World Cup X*. Berlin : Springer Verlag, 2007

[16] WEISSTEIN, E.W.: Least Squares Fitting. In: *MathWorld – A Wolfram Web Resource*. `http://mathworld.wolfram.com/LeastSquaresFitting.html`, Stand: 26.08.2007

[17] WEISSTEIN, E.W.: Line-Line Intersection. In: *MathWorld – A Wolfram Web Resource*. `http://mathworld.wolfram.com/Line-LineIntersection.html`, Stand: 26.08.2007

[18] WEISSTEIN, E.W.: Point-Line Distance – 2-Dimensional. In: *MathWorld – A Wolfram Web Resource*. `http://mathworld.wolfram.com/Point-LineDistance2-Dimensional.html`, Stand: 26.08.2007