Hochschule für Technik, Wirtschaft und Kultur Leipzig
Fakultät Informatik, Mathematik und Naturwissenschaften

Bericht zum Masterprojekt

Entwicklung eines externen, kamerabasierten Trackingsystems für mobile Roboter

Name: Felix Hain
Matrikel-Nr.: 62283
Studiengang: Informatik
Leipzig, den 31. März 2015

Betreuende Professorin:
Prof. Dr. Sibylle Schwarz

Betreuer des Nao-Teams:
M. Sc. Thomas Reinhardt

Inhaltsverzeichnis

1 Aufgabenstellung

1.1 Motivation

Bei den jährlichen RoboCup-Meisterschaften treten Teams aus Robotern im Fußball gegeneinander an. In der Standard Platform League sind dies Roboter des Typs Nao von der Firma Aldebaran Robotics. Diese müssen während eines Fußballspiels komplett autonom agieren, d.h. sie dürfen keine Hilfe durch menschliche Zuschauer erhalten, sondern müssen das Spiel selbst bewältigen. Eine Grundvoraussetzung ist, dass die Roboter mit ihrer eigenen Sensorik die Spielsituation erkennen und die Positionen der auf dem Spielfeld befindlichen Objekte, einschließlich sich selbst, korrekt bestimmen können. Dazu werden auf den von den internen Kameras gelieferten Bildern stationäre Spielfeldmarkierungen, wie Randlinien und Torpfosten, identifiziert und anhand derer die Position des jeweiligen Roboters berechnet. Der vom Nao-Team der HTWK Leipzig verwendete Lokalisierungsalgorithmus befindet sich jedoch in der Entwicklung und weist derzeit große Schwächen in der Genauigkeit auf. Um diese Genauigkeit zu analysieren und Informationen über mögliche Fehler oder Verbesserungsmöglichkeiten zu sammeln, kann eine externe Beurteilung der Roboterposition auf dem Spielfeld nützlich sein.

1.2 Zielsetzung

Ziel des Projektes ist, ein externes System zur Lokalisierung eines oder mehrerer Roboter auf dem Spielfeld zu erstellen. Ein solches System ist in tatsächlichen Wettkämpfen nicht zugelassen, daher kann es fest im Nao-Labor der HTWK installiert werden, wo ein Testspielfeld für die Roboter vorhanden ist. Es sind Kameras über dem Spielfeld zu montieren, die gemeinsam dazu in der Lage sind, die gesamte Spielfeldfläche abzudecken, Markierungen zu entwerfen mit denen die Roboter aus der Sicht dieser Kameras identifizier- und unterscheidbar sind, und eine dazugehörige Software zu schreiben, die in der Lage ist, die Kameras anzusteuern, besagte Identifizierung vorzunehmen, und die tatsächliche Positionierung und Rotation der Roboter, in Spielfeldkoordinaten umgerechnet, auszugeben.

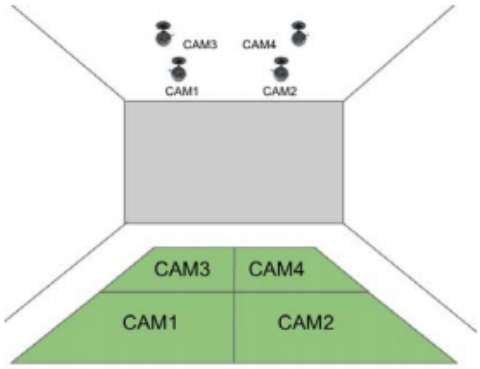

Abbildung 1: Beispielhafte Darstellung der Kamerapositionierung in einem solchen System

Quelle: [1]

1.3 Anforderungen

Für das Ergebnis des Projektes wurden im Hinblick auf oben genannte Zielsetzung von Thomas Reinhardt im Namen des Nao-Teams der HTWK folgende konkrete Anforderungen aufgestellt:

- komplette visuelle Feldabdeckung mit 1, 2 oder 4 Kameras, sodass ein Roboter bis zu 30 cm außerhalb des Spielfeldrandes verfolgt werden kann
- Echtzeitfähigkeit, d.h. die Möglichkeit einer Verarbeitung von mindestens 10 Bildern pro Sekunde auf einem i7@2,0 Ghz
- hohe Genauigkeit der Positionsberechnung auf 3cm oder besser
- hohe Zuverlässigkeit der Markerdetektion mit max. 1% False Positives und min. 95% Trefferrate
- mit der Kodierung der Farbmarkierungsplatten sollten bis zu 12 verschiedene Roboter voneinander unterschieden werden können
- Roboter muss nur in aufrecht stehender Position erkannt werden
- Programmierung der Software entweder in C++ oder in Java

1.4 Teilaufgaben

Um die gestellte Gesamtaufgabe zu bewältigen, wurde sie in eine Anzahl kleinerer Teilaufgaben unterteilt. Diese sind:

- Beschaffung geeigneter Kameras unter der Beachtung der Kriterien Preis und benötigter Anzahl, abhängig vom Öffnungswinkel der Kameras und der Deckenhöhe im Nao-Labor
- programmtechnische Ansteuerung der Kameras, Auslesen des Kamerabildes
- Montage der Kameras an der Labordecke, Verlegung von Kabeln für den Anschluss an einen Rechner im vorderen Teil des Labors
- Implementierung eines Verzeichnungskorrektur-Algorithmus zur Begradigung der Linien im Kamerabild
- Implementierung eines Perspektivkorrektur-Algorithmus zum perspektivischen Entzerren des Kamerabildes
- Bau einer GUI zur Kalibrierung der Perspektivkorrektur und Ausgabe des korrigierten Kamerabildes
- Erstellung eines visuellen Markers der auf dem Kopf eines Naos stabil befestigt werden, später jedoch schnell ausgetauscht bzw. wieder abgenommen werden kann
- Implementierung eines Algorithmus zur Erkennung des Markers
- Umrechnung der Position und des Drehwinkels des dazuehörigen Roboters in Spielfeldkoordinaten unter Berücksichtigung der Kamerakalibrierung
- Integration der Software als Plugin in NaoControl

Zudem wurden folgende optionale/weiterführende Aufgaben gegeben, die aber im Rahmen des Masterprojekts nicht zwigend umgesetzt werden mussten:

- Erkennung mehrerer Roboter
- ID-Zuweisung zu jedem Roboter anhand des visuellen Markercodes
- Zusammenrechnen erkannter Positionen an Kamerabild-Überlappungsbereichen
- Verbesserung des Robotertrackings mittels Kalman-Filter
- lichtunempfindliche Markerdetektion
- automatische Kalibrierung des Kamerabildes durch Erkennung der Spielfeldlinien

2 Lösungsplan

Als erster Schritt der Projektarbeit wurde die Suche nach einem geeigneten Kameramodell in Angriff genommen. Dieses musste von einem Rechner aus ansteuerbar sein und mit möglichst wenigen Kameras das gesamte Spielfeld abdecken können. Eine Anbringung direkt unter der Decke des Labors wurde für am sinnvollsten befunden. Die Kameraansteuerung sollte durch Erstellung eines zunächst simplen Programms getestet werden.
Danach sollte, falls notwendig, eine Verzeichnungskorrektur entworfen und programmiert werden. Dazu erwies sich die Quelle [4] als vielversprechendes Vorbild.
Im Anschluss war eine perspektive Entzerrung des Bildes geplant, um eine orthogonale „Draufsicht“ auf das Feld zu simulieren. Dazu wurden verschiedene Quellen zum Thema Perpektivtransformation mit homogenen Koordinaten konsultiert.
Als Letztes war der Entwurf und die Erkennung von Markern zur Montage auf den Robotern geplant. Marker sollten, inspiriert von der Small Size Leage des RoboCup, aus mehreren verschiedenfarbigen Flecken bestehen, wie beispielsweise in der Quelle [6] beschrieben, die mit Klebe- oder Klettband auf den Robotern befestigt werden sollten. Als Befestigungsort sollte der Kopf dienen, da dieser bei einem aufrechtstehenden Roboter aus der Draufsicht jederzeit sichtbar ist. Die Lokalisierung dieser Marker sollte dann über eine Farberkennung erfolgen.

Die Wahl der Programmiersprache fiel auf Java, da, im Vergleich zu der in den Anforderungen genannten Alternative C++, bessere persönliche Kenntnisse bestanden. Zum Ansprechen der Kameras wurde die Programmbibliothek JavaCV gewählt, welche einen Java-Port der in der Nao-Software verwendeten C++-Bibliothek OpenCV darstellt.

3 Umsetzung

3.1 Kamerabeschaffung und -montage

Als erster Arbeitsschritt mussten geeignete Kameras ausgewählt und besorgt werden. Kriterien für die Auswahl waren primär Öffnungswinkel und Bildauflösung, um mit möglichst wenigen Kameras eine vollständige Spielfeldabdeckung zu gewährleisten, sekundär der Kaufpreis. Eine Abdeckung des gesamten Spielfeldes mit nur einer Kamera stellte sich dabei als unmöglich heraus. Eine Abdeckung mit zwei Kameras bedeutet, dass jede Kamera eine Hälfte des Spielfeldes überblicken könnte, bei einer Abdeckung mit vier Kameras nur ein Viertel. Letzteres wäre ungünstig für die Kamerakalibrierung, da in diesem Fall nicht an jedem der Ränder des jeweils zu überblickenden Bereichs Spielfeldlinien zur Orientierung vorhanden gewesen wären. Daher wurde eine Abdeckung mit zwei Kameras bevorzugt. Die Wahl fiel nach einer Recherche bei verschiedenen Verkäufern auf die Actioncam SJ4000. Zwei Kameras dieses Typs sowie zwei 20 Meter lange USB-Verlängerungskabel wurden beim Onlinehändler Amazon bestellt.

Abbildung 2: erworbene Hardware

Quellen: [2], [3]

Um eine Erkennung von Robotern überall auf dem Spielfeld zu ermöglichen, muss nicht nur das Feld selbst, sondern auch ein gewisser Zusatzbereich im Sichtfeld der Kameras liegen, da ein am Spielfeldrand stehender Roboter bei schräger Betrachtung auf einem zweidimensionalen Kamerabild aus dem Spielfeld „herausragt“:

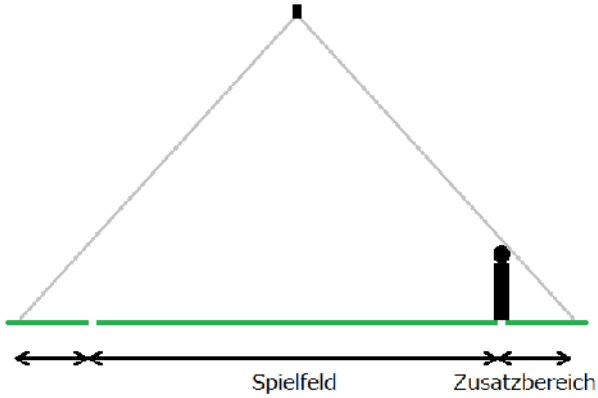

Abbildung 3: ein zusätzlicher Randbereich wird benötigt

Um unter Berücksichtigung der begrenzten Deckenhöhe im Labor von ca. 2,80 m das gesamte 7,5×5 m große Spielfeld mitsamt Zusatzbereich abzudecken, wurden die Kameras nicht senkrecht nach unten, sondern schräg blickend positioniert. Mit Hilfe des Hausmeisters wurde auf beiden Seiten des Feldes ungefähr über dem Rand des Mittelkreises jeweils eine Metallschraube in der Decke befestigt; anschließend wurden die Kameras an diesen so angebracht, dass sie über Kreuz auf die jeweils andere Spielfeldhälfte blicken. Die Verlängerungskabel wurden an den Deckenlampen befestigt und zu den Tischen im vorderen Teil des Labors geführt, wo sie an einen beliebigen Rechner angeschlossen werden können.

Abbildung 4: Positionierung der Kameras an der Labordecke

Abbildung 5: Nahaufnahme einer der Kameras

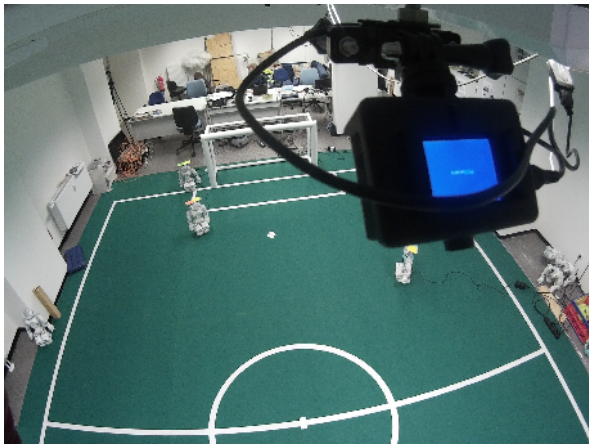

Abbildung 6: ungefährer Blickwinkel der Kamera

3.2 Verzeichnungskorrektur

Das von den Kameras gelieferte Rohbild weist aufgrund des großen Sichtwinkels eine starke tonnenförmige Verzeichnung auf. Dies ist in den Abbildungen 4 und 6, die mit der zweiten Kamera geschossen wurden, gut zu erkennen: Linien, die in der Realität gerade sind, werden im Bild deutlich gekrümmt wiedergegeben. Um später eine vernünftige Positionsberechnung auf dem Feld vornehmen zu können, muss diese Krümmung aus dem Bild entfernt werden.
Dazu wurde ein Verfahren nach dem Vorbild der Quelle [4] angewandt. Um die Verzeichnung auszumessen, wurde ein Testmuster mit einem Gitter gleichmäßig angeordneter horizontaler und vertikaler Linien präpariert.

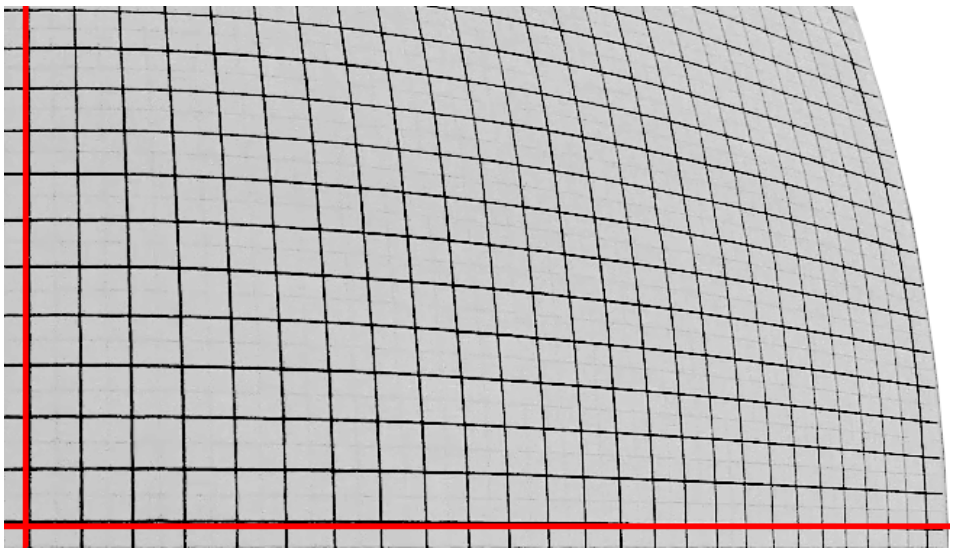

Abbildung 7: Testmuster im 1. Bildquadranten

Die Schnittpunkte der Linien wurden als Referenzpunkte verwendet. Aus dem Verhältnis zwischen den realen Koordinaten eines Punktes auf dem Muster und den verfälschten Koordinaten im aufgenommenen Bild wurde eine Reihe von Faktoren errechnet; diese drücken die Vergrößerungsfaktoren an den entsprechenden Bildpunkten in Abhängigkeit vom Abstand zur Bildmitte aus. Diese Wertezuordnungen wurden anschließend mit dem Computeralgebrasystem Maxima und der Methode der kleinsten Quadrate durch ein Polynom dritten Grades angenähert.
Somit entstand eine Funktion $f(r)=1+a_1 r+a_2 r^2+a_3 r^3$ (mit bekannten Parametern a_1, a_2, a_3), mittels derer aus jedem realen Bildpunkt (x, y) der dazugehörige verfälschte Bildpunkt (x', y') bestimmt werden kann, indem aus den Komponenten x und y der Abstand r des Pixels von der Bildmitte berechnet, in die Funktion eingesetzt, und mit dem Funktionswert $f(r)=r'$ die Komponenten x' und y' berechnet werden.
Um ein verzeichnetes Bild zu korrigieren, wird die Umkehrung dieser Funktion benötigt. Diese wird von der entwickelten Software beim Programmstart automatisch per Bisektion berechnet und ein einem Array gespeichert, sodass die benötigten Werte im laufenden Programm nicht ständig neu berechnet werden müssen. Lediglich die Koordinaten im Verhältnis zur Bildmitte werden für jeden Pixel neu berechnet, der dazugehörige Korrekturfaktor wird aus dem vorberechneten Array entnommen. Auch dies muss nur für einen der Quadranten eines jeden Bildes durchgeführt werden, die restlichen können durch Spiegelung der Koordinaten an den Koordinatenachsen ergänzt werden.

Somit entsteht aus einem tonnenförmig verzeichneten rechteckigen Kamerarohbild ein verzeichnungsfreies Bild in Kissenform.

Abbildung 8: unkorrigiertes Kamerarohbild

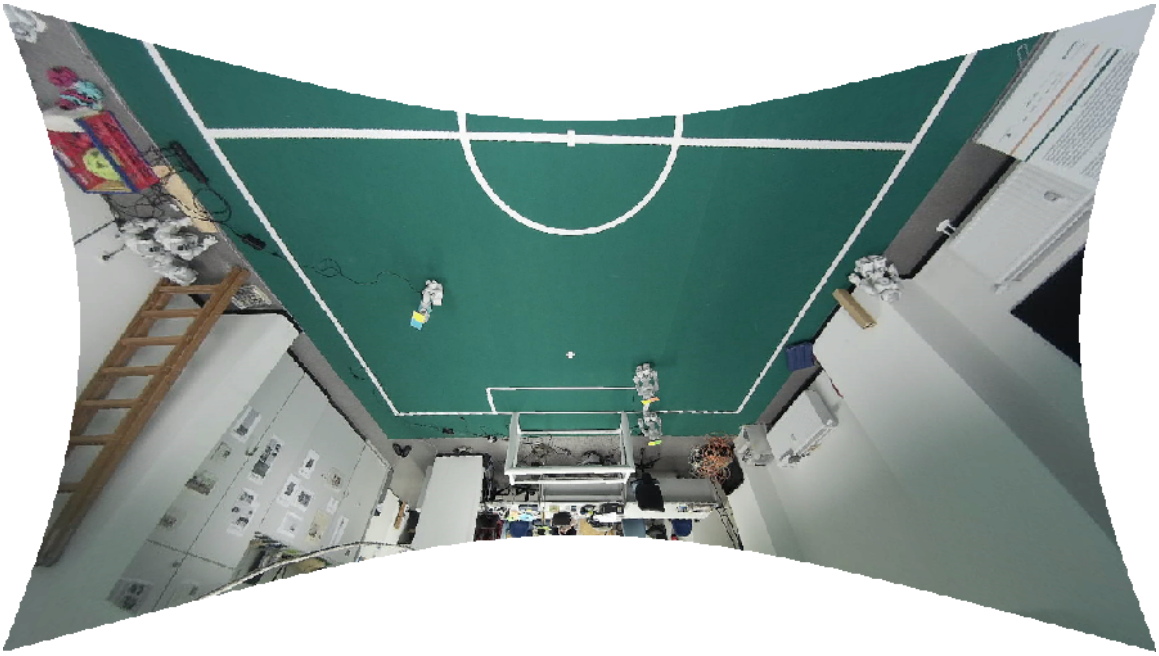

Abbildung 9: korrigiertes Bild

3.3 Perspektivkorrektur

Nach der Verzeichnungskorrektur sind die Feldlinien im Bild begradigt. Allerdings sieht das Feld noch immer nicht wie gewünscht aus. Durch den schrägen Blickwinkel der Kamera wird es stark trapezförmig dargestellt, zudem ist es möglicherweise leicht schräg und versetzt, da sich nicht gewährleisten lässt, dass die Kamera auch perfekt mittig und gerade über dem Feld platziert ist. Diese Probleme werden im Folgenden durch eine Perspektivkorrektur behoben. Die Verzeichnungskorrektur sorgt dafür, dass tatsächlich gerade Feldlinien auch im Bild gerade sind; die Perspektivkorrektur soll dafür sorgen, dass zueinander tatsächlich parallele Feldlinien auch im Bild parallel sind, sowohl zueinander als auch zu den Bildrändern.
Für eine perspektivische Transformation in homogenen Koordinaten ist eine 3×3-Transformationsmatrix nötig. Diese wird mit der JavaCV-Methode
`getPerspectiveTransform(double[] src, double[] dst, CvMat map_matrix)`
aus den beiden Arrays src und dst berechnet. Die Elemente der Arrays stellen die Eckpunktkoordinaten zweier Vierecke dar: src ist ein Quellviereck, dst das Zielviereck auf welches src durch die Transformation abgebildet wird. Über die beiden Vierecke ist die Perspektivkorrektur kalibrierbar. src ist vom Nutzer der Software über die GUI oder programmtechnisch über entsprechende Methoden einstellbar und sollte aus den Eckpunkten der im Bild sichtbaren Spielfeldhälfte gewählt werden. dst wird automatisch errechnet aus den Zielbilddimensionen und einem für jede Bildkante separat einstellbarem Zusatzrand (siehe Kapitel 3.1).
Anschließend wird die Matrix verwendet, um das aus dem vorigen Schritt (Verzeichnungskorrektur) erhaltene Bild zu transformieren.

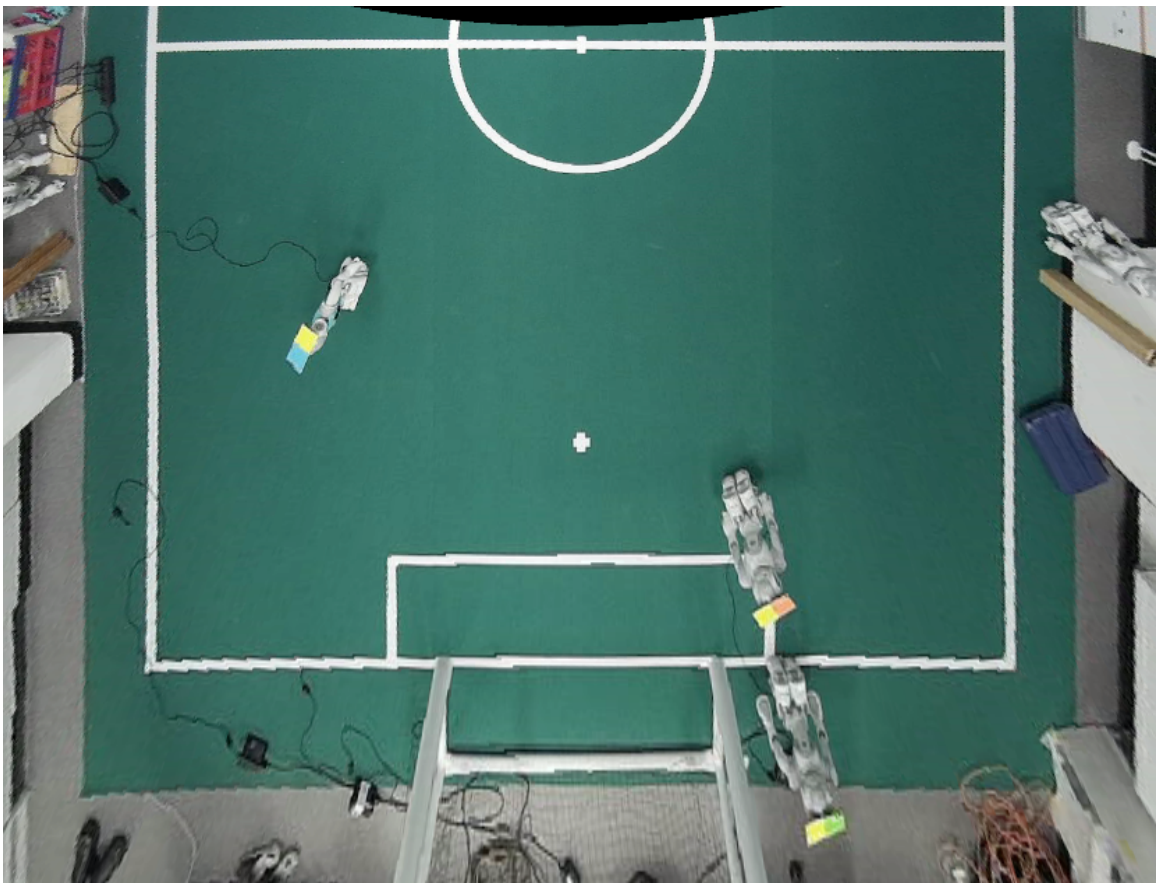

Abbildung 10: transformiertes Bild

3.4 Markerentwurf und -erkennung

Die Kamerabilder werden dem Programm im RGB-Farbsystem geliefert. Bei diesem System werden Farben als Zerlegung in einen Rot-, Grün- und Blauanteil verschiedener Intensität dargestellt. Dies war für die bisherigen Bearbeitungsschritte ausreichend, eignet sich aber nicht besonders gut für eine Markererkennung: hier ist eine intuitivere Farbdarstellung von Nutzen, welche einen Vergleich der Ähnlichkeit verschiedener Farben zueinander erlaubt. Zu diesem Zweck wurde eine Repräsentation des HSL-Farbsystems programmiert, das dem Farben durch die Komponenten Farbton (Hue), Sättigung (Saturation) und Helligkeit (Lightness) beschreibt. Implementiert wurden dazu Konvertierungsfunktionen von RGB nach HSL und umgekehrt sowie eine Funktion zum Vergleich der Ähnlichkeit zweier HSL-Farben nach einer Berechnung aus der Quelle [5]. Die Vergleichsfunktion bietet nun die Möglichkeit, das Bild nach verschiedenen Farben zu durchsuchen. Dabei stellt sich die Frage nach den auf den Markern zu verwendenden Farben. Als Markerfarben bieten sich solche an, die leicht voneinander unterscheidbar sind, aber normalerweise auf dem Spielfeld nicht vorkommen. Da die Feldlinien weiß, einige Objekte am Rand sowie Schatten schwarz (oder zumindest dunkel), die Roboter größtenteils grau und das Spielfeld dunkelgrün ist, sind diese Farben als Markerfarben ungeeignet. Gut geeignete Farben sind hingegen solche, die eine hohe Sättigung und mittlere Helligkeit aufweisen, mit Ausnahme von Rot, da dieses die Farbe des Spielballs ist.
Aus Laufzeitgründen (s. u.) wurde entschieden, dass eine besonders zuverlässig erkennbare Farbe als primäre Markerfarbe verwendet wird, die auf jedem Roboter vorhanden sein muss. Sonstige auf dem Marker vorhandene Farben sind Sekundärfarben und variieren von Roboter zu Roboter. Mit Ihnen werden die Roboter voneinander unterschieden, sowie deren Blickrichtung ermittelt. Für die im Rahmen des Projektes gemachten Tests wurde Gelb als Primärfarbe, Hellblau, Orange und Hellgrün als Sekundärfarben verwendet. Es wurden Marker aus nebeneinander angeordneten 10×10cm großen Quadraten der Primär- und je einer Sekundärfarbe erstellt und mit Klett- und Klebeband auf den Köpfen dreier Roboter befestigt. Diese waren bereits auf den vorigen Abbildungen zu sehen.

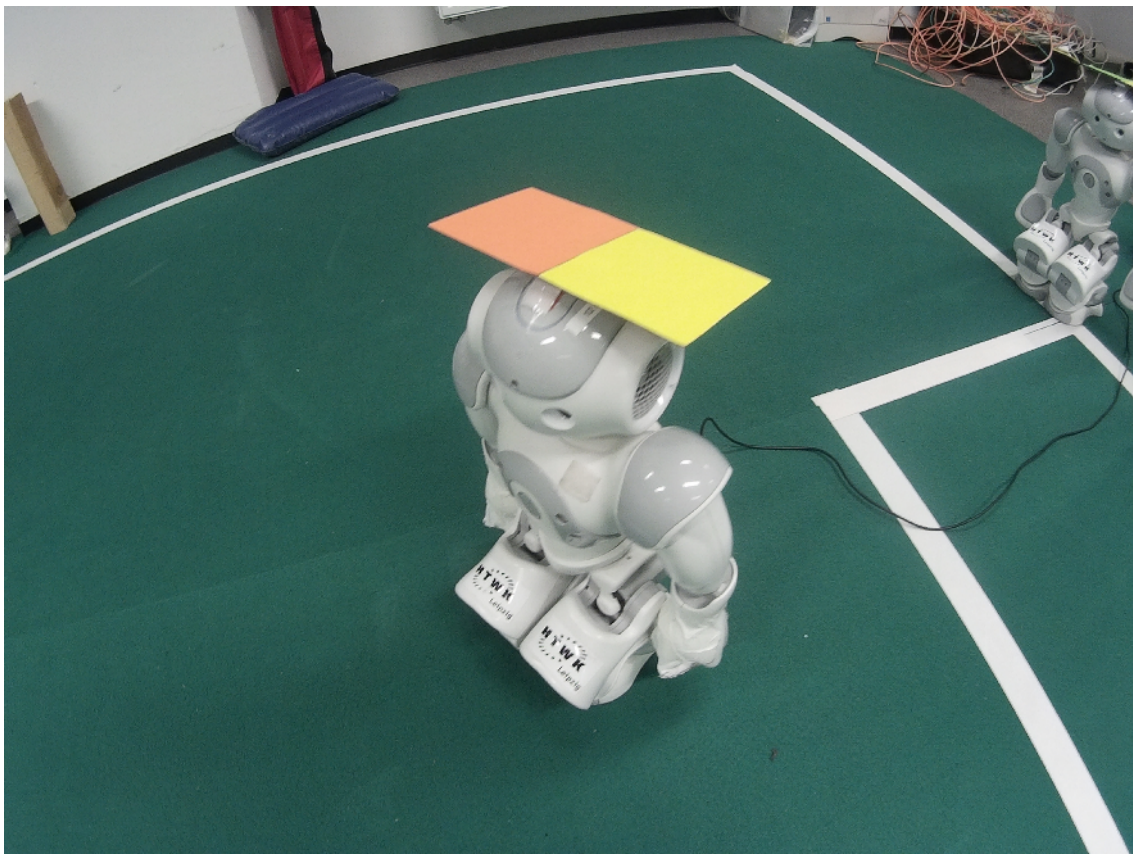

Abbildung 11: Roboter mit gelb/orangefarbenem Marker

Eine simple Suche nach solchen Markern in einem entzerrten Kamerabild wäre nun möglich, indem für jeden Pixel des Bildes dessen HSL-Farbwert bestimmt, und mittels der zuvor implementierten HSL-Vergleichsfunktion mit den gesuchten Markerfarben verglichen wird. Liegt die von der Vergleichsfunktion zurückgelieferte Farbdifferenz unter einem bestimmten Schwellwert, wird der Pixel als der Farbe zugehörig betrachtet anderenfalls nicht. Damit erhielte man pro Farbe ein Binärbild, welches die Position ensprechend gefärbter Pixel im Kamerabild angibt. Da die implementierte HSL-Vergleichsberechnung von trigonometrischen sowie der Wurzelfunktion Gebrauch macht, ist sie jedoch laufzeittechnisch teuer. Die oben beschriebene simple Suchmethode ist aus diesem Grunde nicht echtzeitfähig und kann auf dem zum Testen verwendeten i7-Prozessor bis zu mehrere Sekunden dauern. Durch die bei der Markergestaltung getroffene Entscheidung, eine auf jedem Roboter vertretene Primärfarbe zu verwenden, wurde die Anzahl der im Gesamtbild zu suchenden Farben auf eins reduziert. Die daraus resultierende verringerte Laufzeit von durchschnittlich einer halben Sekunde pro Einzelbild ist dennoch nicht tragbar. Daher wurde ein optimiertes Suchverfahren entwickelt, welches in mehrere sequentielle Schritte unterteilt ist:
Im ersten Schritt wird die gesamte Bildfläche mit einem Raster der Größe n durchsucht, d.h. nur jedes n-te Pixel in horizontaler und vertikaler Richtung wird ausgelesen und mit dem Primärmarkerfarbwert verglichen. Es wird davon ausgegangen, dass bei dieser Suche von jeder entsprechend gefärbten Fläche zumindest ein einziger Pixel gefunden wird. Um jeden so gefundenen Pixel wird anschließend ein Rechteck mit einer Kantenlänge > n gelegt; im Falle von Überschneidungen werden betroffenen Rechtecke gegebenenfalls vergrößert bzw. miteinander vereint.

Als zweiter Schritt werden die so markierten Regionen nochmals durchsucht, diesmal unter Berücksichtigung aller darin enthaltenen Pixel. Dadurch, dass die pixelgenaue Suche nun nur noch in den vorher selektierten Regionen stattfindet, erhöht sich die Rechengeschwindigkeit beträchtlich, sodass die Laufzeit selbst bei mehreren Markern und inklusive der im Folgenden beschriebenen weiteren Schritte nun im niedrigeren zweistelligen Millisekundenbereich liegt.
In einem dritten Schritt wird ein von Thomas Reinhardt bereitgestellter Connected-Components-Labelling-Algorithmus eingesetzt. Dieser bekommt ein Binärbild mit allen im zweiten Schritt gefundenen Pixeln übergeben und trennt diese in voneinander abgegrenzte, nummerierte Bereiche. Bereiche, die in x- oder y-Richtung über oder unter einer festgelegten Größe liegen werden verworfen, alle übrig bleibenden werden als Markerfarbflächen registriert. Somit wird sichergestellt, dass etwa am Rand liegende große farbige Objekte oder einzelne Störpixel im Bild nicht fehlerhaft als Marker erkannt werden.
Da jede der erkannten Primärfarbflächen nun genau einen Roboter repräsentiert, kann die Suche nach den restlichen, sekundären, Markerfarben nun auf das direkte Umfeld dieser Flächen reduziert werden. Die Schritte zwei und drei werden für diese Farben und Flächen entsprechend wiederholt.

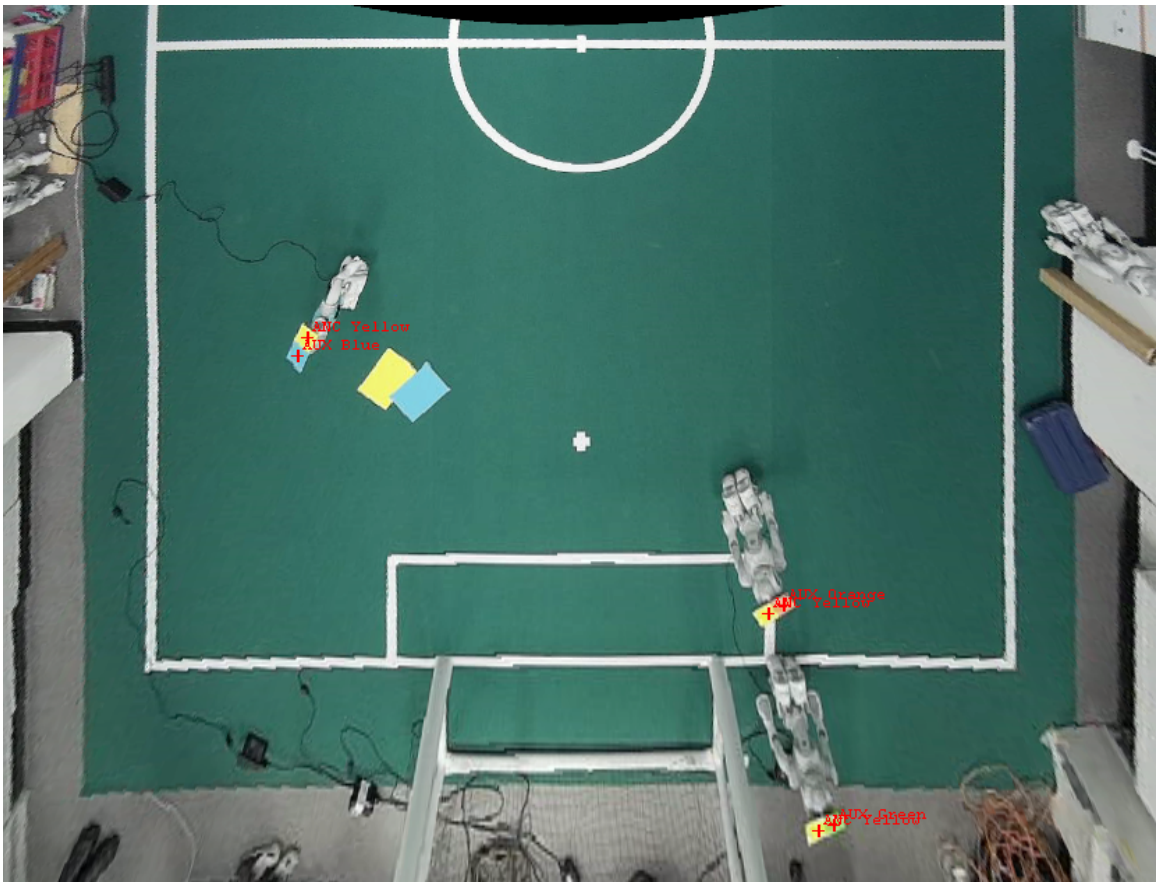

Abbildung 12: Bild mit erkannten Markern

Wie in Abbildung 12 zu erkennen ist, werden die gelben Markierungen mitsamt den blauen, grünen und orangefarbenen um sie herum korrekt erkannt. Die als Beispiel zusätzlich auf dem Spielfeld liegende blau/gelbe Fläche wird ignoriert, da sie aufgrund ihrer Größe kein Marker sein kann.

Nachdem die Markerflächen erkannt sind, kann daraus die Position der Roboter berechnet werden. Die dazu notwendigen Informationen sind, zusätzlich zu den Koordinaten der Marker im Bild:

- die Eckpunkte des Feldes im Bild
- die Eckpunkte des Feldes in Feldkoordinaten
- die Höhe der Marker über dem Boden, sowie
- die exakte Position der Kamera im Raum

Die Bildkoordinaten der Eckpunkte sind aus der Perspektivkorrektur bekannt, die Feldkoordinaten aus der Feldgröße errechenbar und konstant. Die Höhe über dem Boden muss nicht automatisch erkannt werden, da laut Anforderungen nur stehende bzw. laufende Roboter erkannt werden müssen; sie kann allerdings über Parameter konfiguriert werden. Die Raumposition der Kamera wurde ausgemessen und im Programm eingespeichert, sie ist ebenfalls konstant.
Da all diese Informationen also vorliegen, kann mit dem Satz des Pythagoras der Abstand zwischen Kamera und Markern ermittelt werden, sowie die Feldkoordinaten, die die Marker hätten, befänden sie sich auf Spielfeldhöhe. Daraus werden nun für jeden Roboter die Feldkoordinaten des Punktes berechnet, der im Raum senkrecht unter der Mitte seines Markers liegt. Dieser entspricht der Feldposition des Roboters. Die Blickrichtung des Roboters wird dann aus der relativen Position von Primär- und Sekundärfarbe (hier gelb und blau/grün/orange) zueinander berechnet.
Eine Zuordnung von Sekundärmarkerfarben zu Robotern wird verwendet, um die jeweiligen Roboter zu identifizieren.

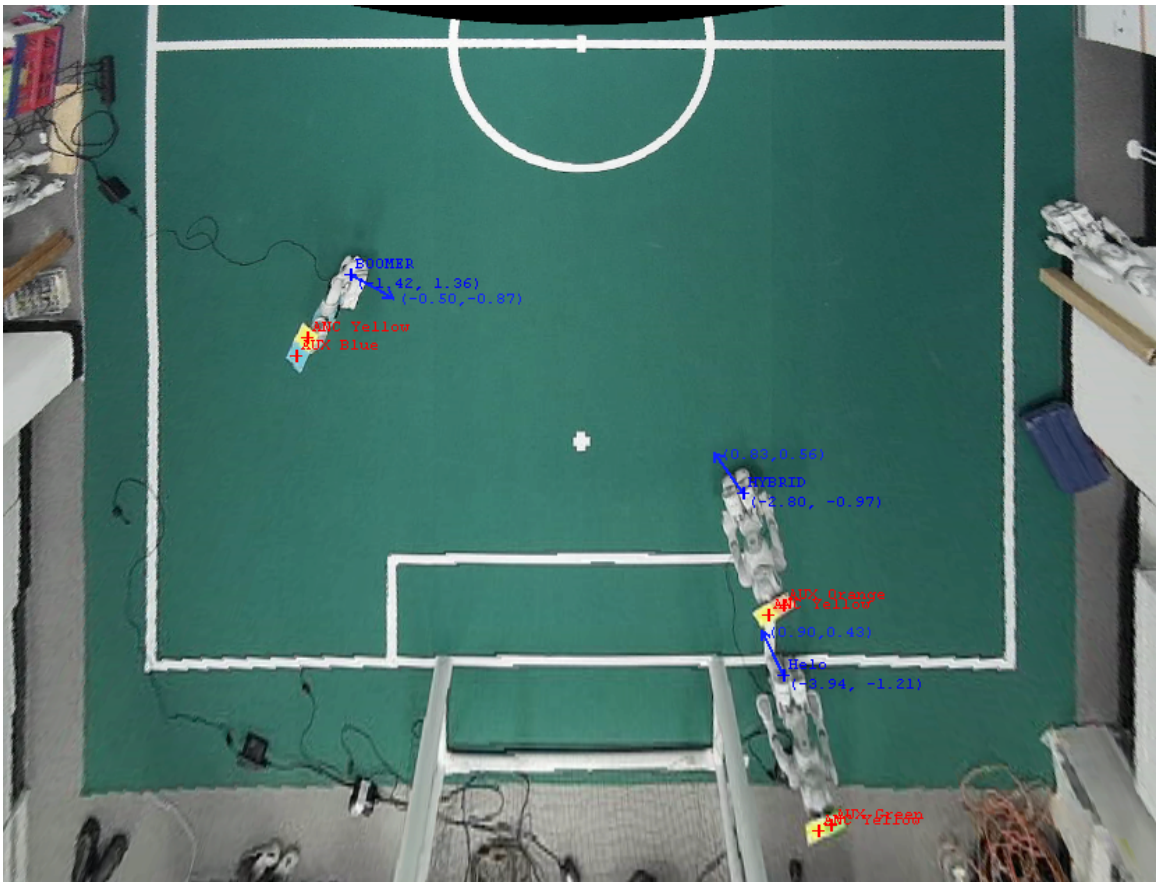

Abbildung 13: Spielfeld mit drei lokalisierten Robotern

4 Programmstruktur

4.1 Gesamtübersicht

Zum Überblick über die Gesamtstruktur der Software wurde ein Klassendiagramm erstellt. Dieses erwies sich zur Einbettung als Bild in dieses Dokument allerdings zu umfangreich und ist daher in der externen Datei Klassendiagramm.png zu finden.
Mit Ausnahme der Klasse `image.LabelImageBinaryLSL` wurden alle dargestellten Klassen selbst und im Rahmen dieses Masterprojekts entwickelt.

4.2 Beschreibung der Pakete

Ergänzend zur im Quellcode enthaltenen Dokumentation sollen hier die Java-Pakete der Software, und einige besondere Klassen, benannt und in ihrer Funktion beschrieben werden.

`program`
Das Paket `program` beinhaltet die Klasse `Program`. Diese enthält die `main`-Methode des Programms. Von hier aus wird eine Kette von `ProcessingUnit`s, sowie eine `ControlPanel`-Instanz gestartet, welche jeweils das Backend und das Frontend des laufenden Programms darstellen. Die Klasse enthält auch eine Methode `shutdown`, mit der das Programm zum Beenden veranlasst werden kann, sowie eine Methode `isRunning`, um abzufragen, ob `shutdown` bereits aufgerufen wurde.
Ebenfalls in diesem Paket enthalten, ist die Klasse `ControlPanel`, die eine GUI zur Bedienung des Programms darstellt. Von hier aus kann der Nutzer zur Laufzeit über eine grafische Benutzeroberfläche die wichtigsten Parameter einstellen, sowie sich live das Kamerabild und die Ergebnisse aller einzelnen Bearbeitungsschritte (beschrieben in Kapitel 3) anzeigen lassen.

`color`
Das `color`-Paket enthält die Klassen `RGB` und `HSL`, welche Farben in den jeweiligen Farbsystemen darstellen und Konvertierungsfunktionen untereinander bereitstellen. Zudem findet sich hier die Enumeration `MarkerColor`, in der die zur Verwendung auf den Robotermarkern vorgesehenen Farben festgelegt sind.

`coords`
Dieses Paket beinhaltet verschiedene Klassen, mit denen Positionen bzw. Koordinaten angegeben werden, darunter mehrere Vektorklassen und die ein Viereck darstellende Klasse `Quad2D`.

`image`
Hier befindet sich die Klasse `CameraImage` sowie die von ihr abgeleiteten Klassen `RectifiedCameraImage` und `AnalyzedCameraImage`, welche als Wrapper für das Kamerabild in verschiedenen Stufen seiner Bearbeitung dienen. RectifiedCameraImage enthält die wichtigen Methoden `toFieldCoords` und `toImageCoords`, mit denen beliebige Bild- und Feldkoordinaten ineinander umgerechnet werden können.

`camera`
Hier enthalten ist die Klasse `Camera`, welche zur Ansteuerung der Kameras verwendet wird.

`processing`
In diesem Paket befinden sich Klassen, die wie in Kapitel 3 beschrieben die Arbeitsschritte Verzeichnungskorrektur, Perspektivkorrektur und Markerdetektion ausführen. Jede dieser Klassen hat eine Reihe von Getter- und Setter-Methoden zur Konfiguration ihrer Parameter sowie eine `run`-Methode, die den entsprechenden Arbeitsschritt auf ein als Parameter übergebenes Einzelbild anwendet.
Dieses Klassen können als API verstanden werden, da es durch das Erstellen von Instanzen dieser Klassen, das Konfigurieren über die Setter und die Übergabe von Bildern an die `run`-Methoden unkompliziert möglich ist, die Kernfunktionen der Software von beliebigem externen Quellcode aus zu nutzen.

`processingunits`
Dieses Paket beinhaltet die größte Anzahl Klassen und ist die programminterne Art, die Funktionen aus dem `processing`-Paket zu nutzen.
Den Kern des Paketes bildet die Klasse `ProcessingUnit`. Diese stellt eine abstrakte Verarbeitungseinheit dar, welche eine Eingabe eines beliebigen, aber bestimmten Datentyps entgegennimmt, in beliebiger Weise bearbeitet, und damit eine Ausgabe eines ebenfalls beliebigen, möglicherweise anderen, Typs produziert. Beliebig viele `ProcessingUnit`s können in einer Baumstruktur aneinandergekettet werden und bilden dann eine Pipeline, in welcher Objekte automatisch von der Ausgabe einer Unit zur Eingabe der jeweils nächsten geleitet werden. Jeder der Units arbeitet dabei unabhängig von den anderen und in einem eigenen Thread, sodass alle Units potentiell parallel arbeiten und die zur Verfügung stehende Prozessorleistung optimal ausnutzen können. ReadWriteLocks, die für die Erschaffung von Synchronisationspunkten zur threadsicheren Übergabe von Daten zwischen den Units notwendig sind, werden automatisch verwaltet. `ProcessingUnit`s können mittels einer `setActive`-Methode zu beliebiger Zeit pausiert und fortgesetzt werden und haben integrierte Funktionen zur Zeitmessung der in ihnen ablaufenden Verarbeitungsschritte.
Durch das Ableiten der Klasse und das Implementieren ihrer abstrakten Methoden wird eine `ProcessingUnit` konkretisiert. Solche konkreten Units gibt es unter anderem für jede der Funktionen aus dem `processing`-Paket, für das Abfragen von Kamerarohbildern, sowie für die Ausgabe von bearbeiteten oder unbearbeiteten Bildern auf dem Bildschirm.
Ein Beispiel für eine Pipeline solcher Units, die auf einem kontinuierlich abgefragten Kamerabild eine Robotererkennung durchführt und das Ergebnis in Echtzeit auf dem Bildschirm ausgibt, wäre:
`CameraUnit → DistortionCorrectionUnit → PerspectiveCorrectionUnit → MarkerDetectionUnit → EnhancedDisplayUnit`

`util`
Ein Paket für Hilfsklassen, die in keins der anderen Pakete einzuordnen sind.

5 Ergebnisse und Ausblick

5.1 Erreichte Ziele

Im Rahmen dieses Masterprojektes wurden fast alle der gestellten Hauptaufgaben erfüllt:

- Kameras wurden erworben und im Labor angebracht
- die Verzeichnung der Kameralinsen wurde gemessen und ein Korrekturalgorithmus implementiert
- die Verzerrung des Feldes im Bild wurde durch eine Perspektivtransformation entfernt um eine rechteckförmige Darstellung zu erreichen
- Marker wurden entworfen und mit Klettband auf den Robotern befestigt
- ein Erkennungsalgorithmus zum Lokalisieren der Marker wurde entworfen und programmiert
- Berechnungen zum Erhalten der Feldposition und Rotation der Roboter aus den Markern wurden implementiert
- eine GUI zum Steuern des laufenden Programms und zur Ausgabe der Ergebnisse wurde gebaut

Zusätzlich wurden mehrere der optionalen Ziele erfüllt:

- mehrere Roboter sind auf dem Feld gleichzeitig erkennbar – getestet wurden bis zu drei, mit weiteren sorgfältig gewählten Sekundärfarben wäre eine noch höhere Anzahl möglich
- eine Bestimmung der Roboteridentität wird mittels einer Zuordnung der Markerfarben zu Robotern erreicht

Mit drei Robotern auf einer Feldhälfte läuft das Programm auf dem zur Entwicklung verwendeten Laptop mit i7-Prozessor mit durchschnittlich 20 Bildern pro Sekunde bei einer CPU-Auslastung von ca. 25%. Ein Hinzuschalten der zweiten Kamera würde aufgrund der Parallelisierung des Programms die CPU-Belastung erhöhen, nicht aber die Bildrate verringern. Das gesteckte Geschwindigkeitsziel ist also erreicht.

5.2 Nicht erreichte Ziele

Eins der ursprünglich gestellten Hauptziele wurde im Rahmen des Projektes nicht umgesetzt:

- Integration der Software als Plugin in NaoControl

Stattdessen wurde allerdings eine Kapselung der Hauptfunktionen in eigene Klassen umgesetzt (Paket `processing`), durch welche die genannten Funktionen leicht aus fremdem Code heraus benutzt werden können und eine Anbindung an NaoControl somit später möglich ist.

Zudem wurden die folgenden optionalen Ziele nicht umgesetzt:

- Zusammenrechnen erkannter Positionen an Kamerabild-Überlappungsbereichen
- Verbesserung des Robotertrackings mittels Kalman-Filter
- lichtunempfindliche Markerdetektion
- automatische Kalibrierung des Kamerabildes durch Erkennung der Spielfeldlinien

Das Zusammenrechnen im Überlappungsbereich konnte aufgrund eines Defektes einer der Kameras im späteren Teil der Bearbeitungszeit nicht mehr realisiert werden, die anderen optionalen Ziele wurden aus zeitlichen Gründen im Rahmen des Masterprojektes nicht mehr umgesetzt.

Gewisse Schwächen bestehen noch in der Zuverlässigkeit der Farberkennung sowie der Haltbarkeit der Marker. Die Farberkennung könnte noch verbessert werden, indem eine lichtunabhängige Markererkennung verwendet würde, denn trotz der guten Ausleuchtung des Labors ist diese nicht perfekt gleichmäßig über dem gesamten Feld. Alternativ könnte ein sehr sorgfältiges Kalibrieren der Markerfarben oder die Verwendung einer alternativen Farbvergleichsfunktion Abhilfe schaffen. Eventuell könnte damit auch eine geringere Markermindestgröße erreicht werden.
Die Befestigung der Marker auf den Robotern mit Klett- oder Klebeband erwies sich als nicht haltbar genug, um einen Sturz des Roboters zu überstehen. Eine Alternative wäre möglicherweise eine Befestigung durch das Öffnen des Roboterkopfes und das Einklemmen von Laschen zwischen den Einzelteilen.
Zudem lassen sich mit den im Rahmen des Projektes angefertigten Marker nicht wie ursprünglich gefordert bis zu 12 Roboter voneinander unterscheiden, da dafür 12 Sekundärfarben zur Verfügung stehen müssten, deren Unterscheidung voneinander schwierig wäre. Die Verwendung von dreiecksförmigen Markern mit je 2 Sekundärfarben würde dieses Problem jedoch beheben da durch die enstehenden Farbkombinationen mehr Roboteridentitäten codiert werden könnten.

Quellen

[1] http://www.mdpi.com/1424-8220/13/11/14954/pdf
[2] http://www.amazon.de/gp/product/B00LL6V1O6?psc=1&redirect=true&ref_=oh_aui_detailpage_o01_s00
[3] http://www.amazon.de/gp/product/B008VDMINU?psc=1&redirect=true&ref_=oh_aui_detailpage_o00_s00
[4] http://robocup.mi.fu-berlin.de/buch/calibration.pdf
[5] http://www.eusflat.org/proceedings/IFSA-EUSFLAT_2009/pdf/tema_0048.pdf
[6] http://www.cs.cmu.edu/~mmv/papers/03icra-jim.pdf

Stand: 30. 03. 2015