Tigers Mannheim

Taktische Spielfeldanalyse im Robocup mittels Spielzugerkennung

Teil B

STUDIENARBEIT
des Studienganges Informationstechnik
an der Dualen Hochschule Baden-Württemberg Mannheim

Oliver Steinbrecher 933748
Paul Birkenkampf 4278912

Mannheim, den 05. Juni 2012

DHBW
Duale Hochschule
Baden-Württemberg
Mannheim

Bearbeitungszeitraum:	26.03.2012 - 05.06.2012
Kurs:	TIT09ANS
Ausbildungsfirma:	Deutsches Zentrum für Luft- und Raumfahrt
Betreuer:	Prof. Dr. Rainer Colgen

Ehrenwörtliche Erklärung

Hiermit versichern wir, dass wir die vorliegende Arbeit selbstständig
und nur unter Benutzung angegebener Quellen und Hilfsmittel angefertigt haben.

Oliver Steinbrecher

Paul Birkenkampf

Mannheim, den 05. Juni 2012

Inhaltsverzeichnis

Abbildungsverzeichnis

Kapitel 1

Abstract

Beobachtungen des Teams Tigers der DHBW Mannheim bei der Roboterfußballweltmeisterschaft im Jahr 2011 und Videoanalysen zeigten, dass selbst die besten Teams der Welt mit relativ simplen Spielzügen immer wieder zum Torerfolg kommen. Besonders der indirekte Schuss wird oft benutzt. Um diese Angriffe abwehren zu können wurde ein Verfahren geschaffen, dass es ermöglicht, indirekte Schüsse zu erkennen. Dafür wurde ein Framebuffer angelegt, mit dem man in die Vergangenheit schauen und analysieren kann, wie es zu einem Torerfolg bzw. -schuss gekommen ist. Mithilfe dieser Daten wurde ein exemplarischer Spielzug geschaffen, der diese Art von Angriffen abwehren kann. Weitere defensive Spielzüge können von den Entwicklern mithilfe dieser Studienarbeit entwickelt werden.

Observations of the team Tigers from the DHBW Mannheim in the robot soccer world championship in 2011 and video analysis showed that even the best teams in the world keep scoring with relatively simple plays. Especially the indirect shot is often used. To fend off these attacks, a process was created that makes it possible to identify indirect shots. For this, a frame buffer has been created, which allows you to look into the past and analyze how a goal or shot on goal has happened. Using this data, an exemplary play were created that can fend off these types of attacks. With this study other defensive plays can be developed by the developers.

Kapitel 2

Einleitung

2.1 Roboterfußball

Mensch gegen Computer - Dieser Wettkampf fasziniert die Menschen schon seit den ersten Schritten in der Informationstechnik. Zuerst nur visuell im Spiel Pong, später dann intellektuell im Schach und in naher Zukunft auch physisch in einer der beliebtesten Sportart des Menschen - Fußball.
Beim RoboCup treffen sich Menschen aus aller Welt, um ihre selbstentwickelten Fußballroboter gegeneinander antreten zu lassen und damit auf spielerische Art immer neue Fortschritte in verschiedenen Bereichen, wie z.B. der Elektrotechnik, Bildverarbeitung oder Künstlichen Intelligenz zu entwickeln, testen und vorzuführen. Dabei verfolgen die Entwickler ein hochgestecktes Ziel:

> *"bis zum Jahre 2050 ein Team von autonomen, humanoiden Robotern zu entwickeln, das in der Lage ist, den zu diesem Zeitpunkt amtierenden menschlichen Fußballweltmeister schlagen zu können."* [1]

Der RoboCup und die Weltmeisterschaft der Federation of International Robot-Soccer Association (FIRA) sind die wichtigsten internationalen Wettkämpfe im Roboterfußball. Beide werden finanziell zum größten Teil von Sponsoren getragen ohne die Roboterfußball nicht möglich wäre.
Beim RoboCup gibt es sechs verschiedene Ligen in denen Fußball gespielt wird, sowie zwei die sich nicht mit Fußball beschäftigen. Diese beiden Ligen sind die Rescue League, die sich mit der Entwicklung von Rettungsrobotern beschäftigt und die @Home League, die Roboter als Haushaltshilfen entwickelt. Die Hauptaufmerksamkeit beim RoboCup liegt jedoch auf den Fußballligen. Diese unterscheiden sich nach den verwendeten Robotern (u.a. auch humanoide Roboter verschiedener Größen). Dabei ist das dynamischste Spiel bei den Robotern in der Small Size League (siehe Abbildung 2.1) zu beobachten.

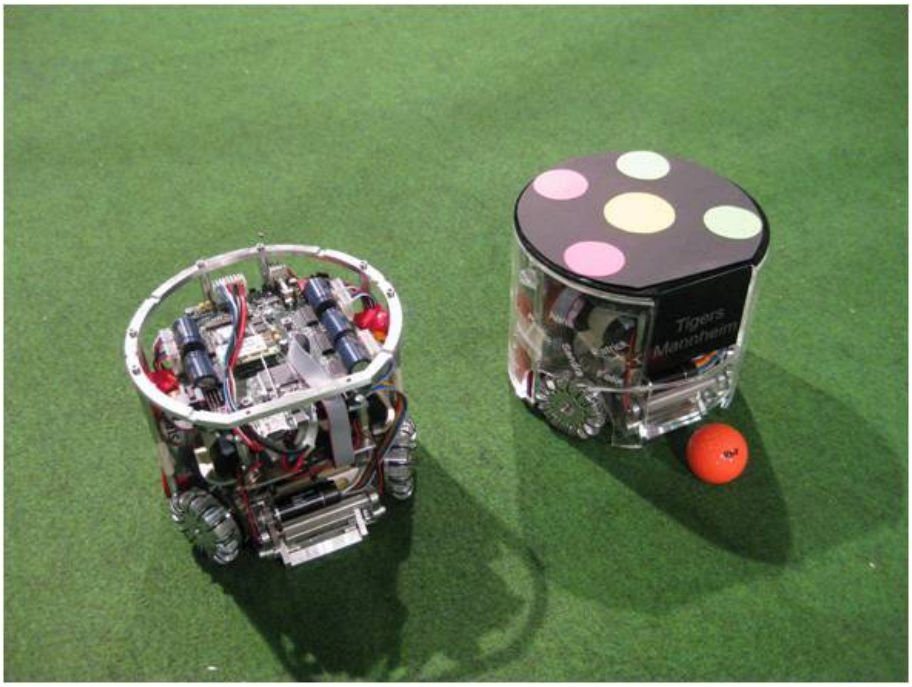

Abbildung 2.1: Zwei Roboter der Small Size League der Tigers Mannheim

2.2 Das Team Tigers Mannheim

Die Tigers (**T**eam **I**nteracting and **G**ame **E**volving **R**obot**S**) der DHBW Mannheim sind ein Team das in der Small Size League aktiv ist. Sie nahmen im Jahr 2011, nach dreijähriger Projektarbeit, erstmals an der Roboterfußballweltmeisterschaft in Istanbul teil. Dort wurden sie inoffiziell als bester Newcomer gehandelt, da sie ein Spiel gewannen und kein Spiel 10:0 verloren (sofortiger Spielabbruch). Im Gegensatz zu vielen anderen Teams bestehen die Tigers Mannheim nur aus Studenten. Die betreuenden Professoren sind nicht aktiv an der Entwicklung des Projekts beteiligt, stehen aber beratend und unterstützend zur Seite.

2.3 Spielmuster/Spielsituation

Bei der Teilnahme am RoboCup 2011 in Istanbul und nach der Studie mehrerer Videos des Turniers und vergangener Jahre wurde festgestellt, dass oft Tore nach dem selben Muster geschossen wurden. Diese Spielmuster sind dabei fast nie komplex aufgebaut sondern relativ simpel. Selbst die besten Teams bedienen sich dieser simplen Spielmuster, können diese aber viel schneller und präziser durchführen als andere. Dadurch konnten sie sich in den vergangenen Jahren von anderen Teams absetzen. Besonders oft führte ein indirekter Schuss zum Erfolg. Dieser indirekte Schuss zeichnet sich dadurch aus, dass direkt vor dem

Torschuss ein Pass zum Schützen durchgeführt wurde (siehe Abbildung 2.2). Könnte man dieses Spielmuster abwehren, z.B. durch Abfangen des Passes oder Blocken des Schusses, könnte man viele Gegentore vermeiden. Daher widmet sich diese Studienarbeit der Erkennung solcher Spielmuster um geeignete Abwehrmaßnahmen ergreifen zu können.

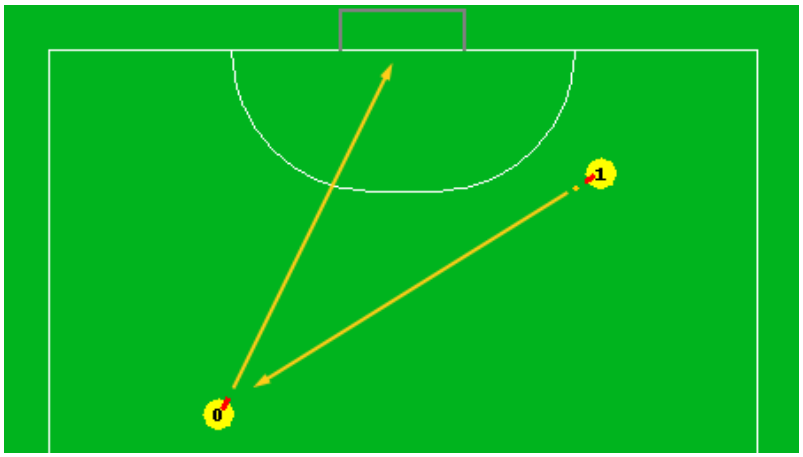

Abbildung 2.2: Definition eines Spielmusters

2.4 Ziele und Anforderungen

Um die eigene Defensivarbeit zu verbessern, soll in dieser Studienarbeit ein Verfahren implementiert werden, welches die in Kapitel 2.3 beschriebenen indirekten Schüsse erkennen kann. Dieses Verfahren soll innerhalb der Zentralsoftware Sumatra entwickelt werden und die Effizienz des Gesamtsystems nicht verringern. Wurde ein Spielmuster erkannt, so soll es in der grafischen Oberfläche visualisierbar sein sodass der Entwickler eine Übersicht über vorhandene Spielmuster hat. Weiterhin soll aufgrund dieser Daten ein exemplarischer Spielzug entwickelt werden, der die erkannten Spielmuster abwehren kann.

Kapitel 3

Theoretische Grundlagen

Das folgende Kapitel beschäftigt sich mit den theoretischen Grundlagen, die dazu verwendet werden, um Spielmuster in einer Datenmenge zu erkennen. Es werden zunächst bedeutsame Begriffe angesprochen und darauf aufbauend Lernkonzepte thematisiert, die eine Klassifizierung und Bewertung einer gegebenen Spielsituation ermöglichen. Dabei steht besonders das instanzbasierte Lernen im Vordergrund.

3.1 Wichtige Begriffe

Bevor nun in den folgenden Kapiteln detailliert beschrieben wird, wie genau einzelne Spielmuster erkannt werden sollen, sollte zunächst verdeutlicht werden, was für Eingaben zur Analyse und Erkennung verwendet werden. Dabei sollen zunächst Grundlegende Begriffe aus dem Bereich maschinelles Lernen erläutert werden.
Unter einer *Konzeptbeschreibung* versteht man das, was das System lernen soll, also die Erkennung, ob die aktuelle Situation einem indirekten Schuss auf das eigene Tor aus der Vergangenheit gleicht. Die Informationen, die ein Lernverfahren dazu verwenden soll, um Entscheidungen zu treffen werden als *Instanzmengen* bezeichnet. Jede *Instanz* kennzeichnet dabei eine spezifische Situation. Dies wäre beispielsweise eine charakteristische Aufstellung der gegnerischen Roboter auf dem Spielfeld zum Durchführen eines indirekten Torschusses. Eine *Instanz* wird durch die Werte von Attributen charakterisiert, die unterschiedliche Aspekte beschreiben können. Grundsätzlich wird bei den Attributen zwischen zwei Typen unterschieden. Dazu gehören u.a., numerische und nominale (oder kategorische) Attribute. Da für die Erkennung von Spielmustern die Position der einzelnen Roboter von Bedeutung ist, werden hauptsächlich nur numerische Attribute, wie die x- und y-Koordinate der Position und die Ausrichtung berücksichtigt.

3.2 Instanzmenge

Innerhalb von *Sumatra*, der zentralen Software zur Steuerung der Roboter, wird pro Kamerabild ein Datenframe erzeugt, der exakte Informationen über alle Roboter und dem Ball auf dem Spielfeld umfasst. Für die Umsetzung der in 2.4 angesprochenen Anforderungen ist die Position der Roboter und deren Ausrichtung, sowie die Position des Balls von größerer Bedeutung. Diese Daten stehen *Sumatra* mit einer Aktualisierungsrate von ca. 60Hz zur Verfügung.
Dieser Datenframe entspricht einer *Instanz*, die als Grundlage für das Lernen eines Konzepts verwendet wird. Wichtig dabei ist, dass eine *Instanzmenge* eine zeitliche Abfolge von *Instanzen* wiederspiegelt und das Lernverfahren immer nur die aktuellste *Instanz*, also den neusten Datenframe, betrachten sollte, um optimale Ergebnisse liefern zu können.

3.3 Lernen eines Konzept

In der Literatur [2] wird zwischen vier unterschiedlichen Verfahren differenziert, die sich mit dem Lernen von *Konzepten* befassen. Beim *klassifizierenden Lernen* nimmt ein Lernverfahren eine Menge klassifizierter Beispiele entgegen. Diese werden verwendet um neue unbekannte Beispiele entsprechend den Klassen einzuordnen. Neben diesem Verfahren existieren noch das *assoziierende Lernen*, das *Clustering* und die *numerische Vorhersage*, jedoch werden diese im Verlauf der weiteren Arbeit nicht betrachtet. Der Grund dafür ist, dass die aktuelle Spielsituationen mittels, in der Vergangenheit liegenden, indirekten Schüssen klassifiziert werden soll, um ermitteln zu können, ob das Tor besonders zu schützen ist.
Unabhängig vom verwendeten Verfahren kennzeichnet ein *Konzept* den zu erlernenden Kontext, während ein Lernverfahren als Ausgabe eine Konzeptbeschreibung liefert. Bezogen auf den gegebenen Anwendungsbereich würde das Verfahren zum klassifizierenden Lernen eine Aussage darüber treffen, welcher Klasse eine unbekannte Testinstanz zugeordnet werden kann.

3.3.1 Gewinnung von Klassen zur Klassifizierung

Unter einer *Klasse* wird in diesem Kontext eine Instanz verstanden die eine, für einen indirekten Torschuss charakteristische, Aufstellung der Roboter beschreibt. Dabei ist vor allem die Position des passenden und dem aufs Tor schießenden Roboter von großer Bedeutung. Diese Klassen werden durch das in Kapitel 2.3 beschriebene Verfahren gewonnen, indem die jeweilige aktuelle Instanz daraufhin analysiert wird, ob sich der Ball mit einer Mindestgeschwindigkeit auf das Tor bewegt. Ist dies der Fall wird aus den vergangenen Instanzen diejenige ermittelt die Abschuss des Balls enthält.
Die Menge der gesammelten Klassen wird dann dazu verwendet, um neue unbekannte Instanzen einer der bisherigen zuzuordnen.

3.3.2 Instanzbasiertes Lernen

Das instanzbasierte Lernen arbeitet mit einer Menge von bereits klassifizierten Testbeispielen, die vorab oder im laufenden Betrieb erzeugt werden. Eine Distanzfunktion wird dann dazu verwendet um festzustellen, welcher der Testklassen einer unbekannten Testinstanz am nächsten kommt. Dazu wird eine Distanzfunktion benötigt, die einen Vergleich zwischen einer Klasse und einer unbekannten Instanz ermöglicht.

3.3.3 Die Distanzfunktion

Die meisten instanzbasierten Verfahren verwenden die im Folgenden näher beschriebene euklidische Distanz. Hierbei wird die Distanz zwischen einer Instanz mit den Attributen $a_1^{(1)}, a_2^{(1)}, .., a_k^{(1)}$ und einer anderen Instanz mit den Parametern $a_1^{(2)}, a_2^{(2)}, .., a_k^{(2)}$ wie folgt mit der Berechnung 3.2 angegeben.

$$\sqrt{(a_1^{(1)} - a_1^{(2)})^2 + (a_2^{(1)} - a_2^{(2)})^2 + ... + (a_k^{(1)} - a_k^{(2)})^2} \tag{3.1}$$

Hierbei ist zu beachten, dass für den Vergleich von Instanzen die Quadratwurzel nicht berechnet werden muss und die Summe der Quadrate direkt zur Klassifizierung verwendet werden kann. Alternativ können auch andere Berechnungsvorschriften verwendet werden, jedoch bietet die euklidische Distanz einen guten Kompromiss zwischen dem Einfluss von großen und kleinen Differenzen dar.
Zusätzlich muss beachtet werden, dass Attribute mit unterschiedlichsten Maßstäben angegeben werden können, wodurch gewisse Attribute einen zu hohen Einfluss auf die Gesamtdistanz erlangen könnten. Aus diesem Grund ist es ratsam alle numerischen Attributwerte so zu normalisieren, dass die in einem Wertebereich von 0 und 1 liegen. Sollten einzelne Attribute einen besonderen Einfluss auf die Gesamtdistanz zweier Instanzen haben so besteht die zusätzliche Möglichkeit diese mit Gewichtungskoeffizienten $w_1, w_2, .., w_k$ zu versehen:

$$\sqrt{w_1 * (a_1^{(1)} - a_1^{(2)})^2 + w_2 * (a_2^{(1)} - a_2^{(2)})^2 + ... + w_k * (a_k^{(1)} - a_k^{(2)})^2} \tag{3.2}$$

Kapitel 4

Implementierung

Das folgende Kapitel legt den Schwerpunkt auf die Umsetzung der in vorherigen Abschnitten angesprochenen theoretischen Grundlagen und Konzepte zur Realisierung der gewünschten Anforderungen. Zunächst wird der strukturelle Aufbau des Systems angesprochen und anschließend der Muster-Klasse und die Implementierung der Distanzfunktion behandelt. Darauf aufbauend werden weitere Funktionalitäten, wie die Persistierung gewonnener Muster und die Erkennung von Torschüssen thematisiert.

4.1 Struktureller Aufbau

Grundsätzlich besteht das System zur Erkennung von Spielmustern aus zwei Teilen mit getrennten Aufgaben, wobei dieser schematisch in Abbildung 4.1 illustriert wird. Das erste Teilsystem, hier blau dargestellt, behandelt ausschließlich die Bewertung der jeweiligen aktuellen Instanz mit den vorliegenden Klassen, wobei wie bereit im theoretischen Teil angesprochen ein instanzbasiertes Verfahren zum Einsatz kommt.

Ergänzend dazu behandelt die zweite Komponente die Erkennung bzw. Gewinnung von neuen Klassen, die dann wiederum zur Analyse der jeweiligen Instanz verwendet werden.

4.2 Einordnung in die Zentralsoftware Sumatra

Sumatra ist die Zentralsoftware der Tigers Mannheim. Sie besteht aus verschieden Modulen die jeweils verschiedene Aufgaben haben. Das Cam-Modul wertet die von den Kameras kommenden Daten aus und schickt diese als CamFrame an das WorldPredictor-Modul. Dieses Modul versucht vorauszusagen wie die Spielsituation nach dem Durchlauf aller Module aussieht. Diese Information wird dann als WorldFrame an das AI-Modul weitergegeben. Hier entscheidet die künstliche Intelligenz unter Berücksichtigung des WorldFra-

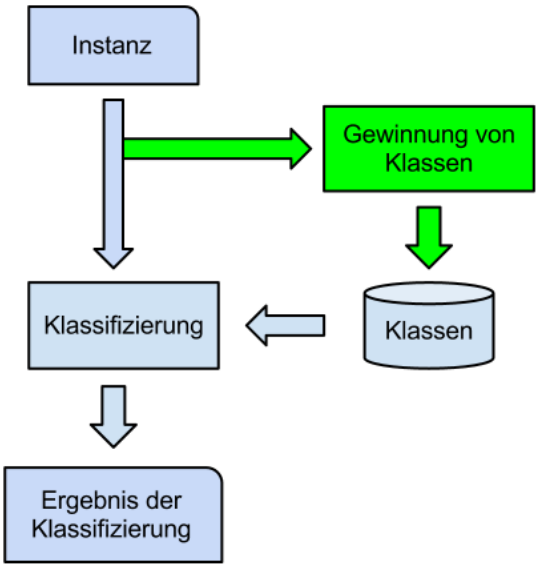

Abbildung 4.1: Zweiteilung der Mustererkennung.

mes und der Schiedsrichterentscheidungen was die Roboter als Gemeinschaft für Spielzüge anwenden sollen. Um diese gemeinsame Bewegung zu ermöglichen, erhält jeder Roboter einen so genannten Skill der vom SkillSystem-Modul in, für den Roboter verständliche, BotCommands umgewandelt wird. Diese BotCommands werden dann vom Botmanager-Modul an die Roboter gesendet die diese Befehle ausführen. Dieser Ablauf ist in Abbildung 4.2 visualisiert. Der gesamte Durchlauf (im folgenden Frame genannt) wird immer wieder wiederholt und dauert beim aktuellen Stand ca. 10ms.

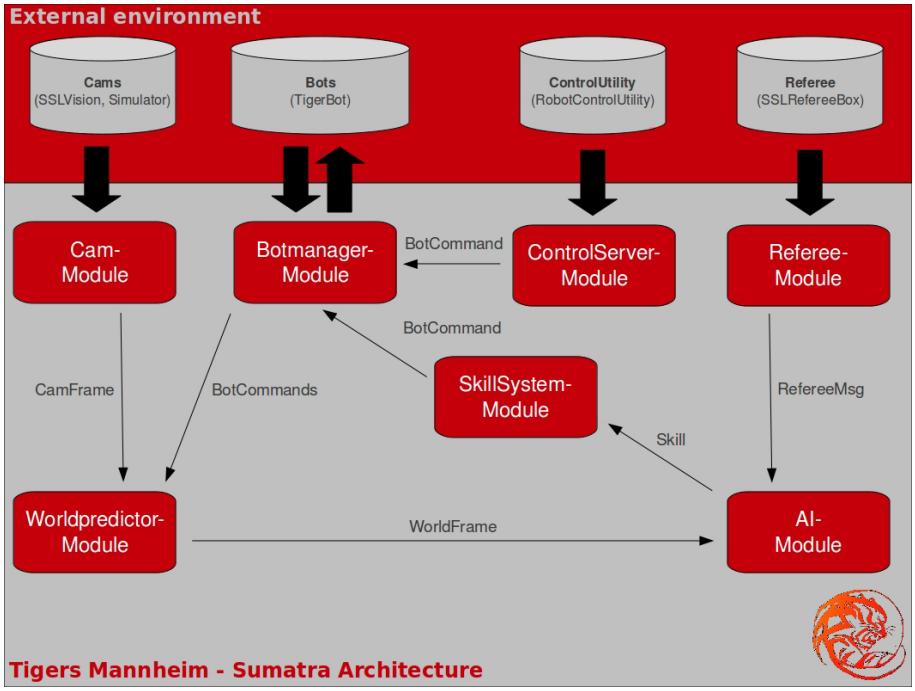

Abbildung 4.2: Modularer Aufbau von Sumatra

Welche Taktiken und Spielzüge im Spiel angewendet werden, wird im AI-Modul entschieden. Dafür gibt es verschiedene Metriken die auswerten, ob das eigene Team oder die Gegner in Ballbesitz sind - also eine Offensiv- oder Defensivtaktik angewendet wird - und welcher Spielzug am erfolgversprechendsten ist. Wurde so ein Spielzug ausgewählt, verteilt das AI-Modul die verschiedenen Rollen (Torhüter, Verteidiger, Passempfänger, usw.) an die einzelnen Bots und weist ihnen Skills (Fahren, Zielen, Schießen, usw.) zu. Neben diesen Entscheidungen erfolgt im AI-Modul auch die Pfadplanung. Hier wird sichergestellt das der Bot seine Zielposition erreicht ohne gegen andere Bots oder, wenn gewünscht, den Ball zu fahren. Da die Studienarbeit bei der Spielzugauswahl behilflich sein soll wird sie im AI-Modul entwickelt.

4.3 Framebuffer

Zur Erkennung eines indirekten Schusses müssen in der Vergangenheit liegende Datenframes ausgewertet werden können. Dafür wurde ein Framebuffer geschrieben der Datenframes aufnehmen kann. Der Framebuffer wird nach dem First In First Out (FIFO) Prinzip gefüllt bzw. geleert. Es wird also der Frame, der am weitesten in der Vergangenheit liegt gelöscht und von einem neuen ersetzt. So kann man dem Framebuffer eine feste Anzahl an Frames geben die er speichern soll. Die Größe des Framebuffer muss dabei so gewählt werden das man einerseits weit genug in die Vergangenheit schauen kann, andererseits aber

auch die Datenmenge nicht zu groß wird. Nach empirischen Untersuchungen stellte sich eine Framebuffergröße von 375 Frames als optimal heraus. Damit kann man 6 Sekunden ($375 * 16ms = 6s$) in der Zeit zurückgehen und die angesammelte Datenmenge fällt nicht ins Gewicht.

4.4 Erkennung von Spielmustern

Zur Erkennung eines Musters müssen verschiedene Bedingungen erfüllt sein die im folgenden dargestellt werden. Grob lässt sich die Erkennung eines Musters in folgende Teilbereiche gliedern: Erkennung des Torschuss, Bestimmung des Schützen und Bestimmung des Passers.
Als erstes muss ein Schuss aufs Tor detektiert werden. Dafür wird aus dem Datenframe zuerst die Geschwindigkeit des Balls bestimmt. Beträgt diese mehr als $1m/s$, könnte der Ball von einem gegnerischen Roboter geschossen worden sein. Danach wird überprüft ob es sich um einen Torschuss auf unser Tor handelt. Dafür wird der Schnittpunkt des Bewegungsvektors des Balls aus dem Datenframe und unserer Torlinie bestimmt. Befindet sich dieser Schnittpunkt innerhalb unseres Tores oder bis zu der Hälfte der Torbreite daneben (siehe rote Linie in Bild 4.3), wird überprüft ob der Bewegungsvektor des Balls auch in Richtung unseres Tores zeigt. Erfüllt eine Situation diese Kriterien wird dies als Torschuss gewertet.

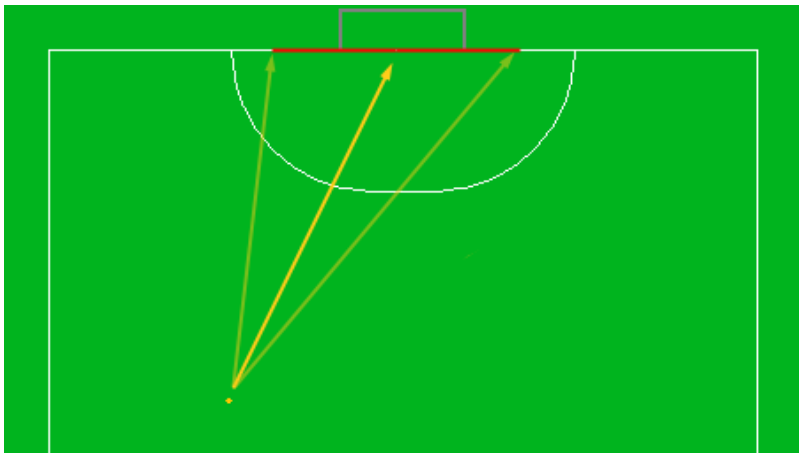

Abbildung 4.3: Definition eines Torschuss

Unmittelbar nach der Erkennung eines Torschusses auf unser Tor wird zuerst der Schütze bestimmt. Dies gestaltet sich relativ simpel. Da der schießende Roboter den Ball sehr schnell stark beschleunigt und dieser die Grenzgeschwindigkeit von $1m/s$ im Prinzip sofort überschreitet, ist der Roboter der dem Ball in diesem Moment am nächsten steht der Schütze. Um den nächsten Roboter zu bestimmen, wird über jeden gegnerischen Roboter

iteriert und die euklidische Distanz bestimmt. Der Roboter mit dem kleinsten Wert ist dann somit der Schütze.
Die Erkennung des Passers gestaltet sich etwas komplizierter. Hierbei muss mithilfe des Framebuffers (siehe Kapitel 4.3) in die Vergangenheit geschaut werden. Der Inhalt des Framebuffers wird hierbei Frame für Frame zurückgegangen und überprüft ob ein Passer vorhanden ist. Ein Passer ist dann vorhanden wenn ein weiterer gegnerischer Roboter (nicht der Schütze) im betrachteten Frame in der Nähe des Balls (max. Abstand: zweifacher Roboterradius) ist. Dafür wird wiederum über alle gegnerischen Roboter iteriert, die euklidische Distanz bestimmt und ausgewertet ob diese kleiner als der zweifache Roboterradius ist. Allerdings wird der bereits bestimmte Schütze aus dieser Berechnung herausgenommen.
Wurden alle bisher beschriebenen Kriterien erfüllt, so ist ein Spielmuster erkannt und die Position des Passers und Schützen gespeichert. Schlägt jedoch nur eine einzige Überprüfung fehl, egal an welcher Stelle, so wird kein Muster erkannt da dies laut Definition aus 2.3 kein Spielmuster sein kann.
Bereits erkannte Spielmuster sollen nicht erneut gespeichert werden da dies nur zu redundanten und somit unnötigen Daten führt. Daher wird nach der Erkennung des Spielmusters erst noch überprüft ob dieses bereits vorhanden ist. Dafür wird die euklidische Distanz zwischen den Positionen (jeweils für Passer und Schützen)des aktuellen Musters und den Positionen in den bereits vorhandenen Mustern berechnet. Sind beide Distanzen kleiner als der dreifache Roboterradius, so wird das Muster nicht abgespeichert da es bereits ein annähernd gleiches gibt.

4.5 Speichern eines Musters

Die Java-Klasse *Pattern* repräsentiert eine Klasse[1], die mit einer unbekannten Testinstanz verglichen und somit zur Klassifizierung verwendet werden kann. Um diese Anforderungen zu erfüllen gibt es verschiedene Methoden, die beispielsweise den Vergleich mittels einer Distanzfunktion umsetzen. Instanziiert wird eine solches Muster mittels der Positionen des passenden und dem aufs Tor schießenden Roboters. Zusätzlich werden bei der Konstruktion des Objekts die aktuellen Geometrie-Informationen über das Spielfeld aus den Konfigurationsdateien von *Sumatra* gelesen damit die Methoden zur Normalisierung der Positionen auf dem Spielfeld korrekt arbeiten können. Diese Normalisierung ist, wie bereits in Kapitel 3.3.3 angesprochen, notwendig um eine ungleiche Gewichtung von Attributen zu vermeiden.

Die Implementierung der Distanzfunktion betrachtet zwei verschiedene Aspekte. Zunächst wird geprüft, ob die aktuelle Roboteraufstellung der Testinstanz dem in der Klasse gespeichertem Muster gleicht und der gegnerische Roboter ermittelt, der der Position des passenden Spielers am nächsten steht. Anschließend wird aus der Menge der verbleibenden Gegnern der bestimmt, der dem aufs Tor schießendem Roboter am nächsten ist. Im

[1] im Kontext von Lernkonzepten (Vgl. 3.3)

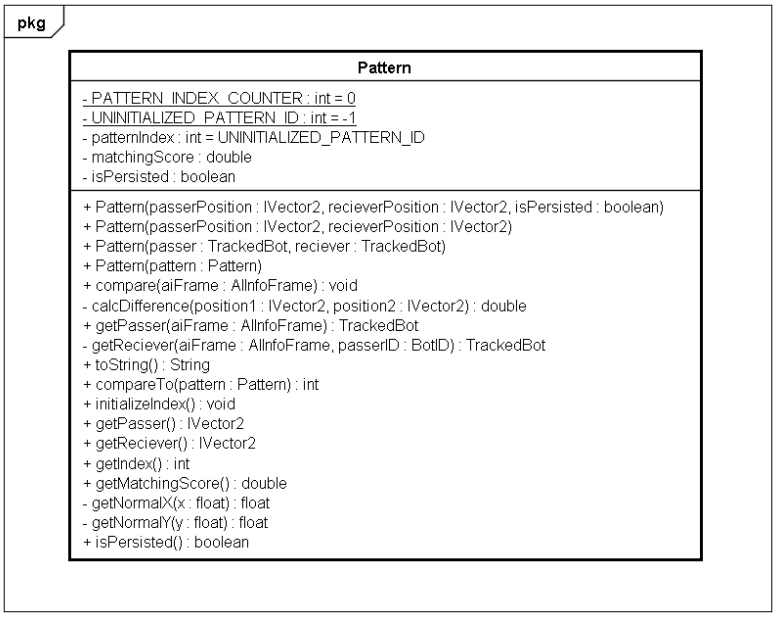

Abbildung 4.4: Darstellung der Klasse *Pattern*

Anschluss daran wird nun die eigentliche Distanzfunktion angewendet. Dabei erfolgt die Berechnung dem in Abschnitt 3.3.3 angesprochenen Verfahren und ist in der folgenden Gleichung 4.1 dargestellt.

$$difference = SCALEFACTOR * [(x_{pattern} - x_{actual})^2 + (y_{pattern} - y_{actual})^2] \quad (4.1)$$

Dabei werden die Berechnungen für beide im Muster abgespeicherten Positionen durchgeführt. Um eine bessere Klassifizierung der Instanz zu ermöglichen wird zudem noch die Lage des Balls auf dem Spielfeld betrachtet. Ein Pattern entspricht dann der aktuellen Instanz, wenn sich auch der Ball in der Nähe des passenden Roboters im Muster befindet. Aus diesem Grund wird die Distanzfunktion auf die Musterposition des 'Passers' in Kombination mit der Ballposition angewendet.

$$score = 1 - (difference_{bot} + difference_{ball}) \quad (4.2)$$

Mit den vorherigen Berechnungen und der Gleichung 4.2 liefert die Methode *compare*(..) als Rückgabewert einen Faktor[2] der angibt, wie gut die aktuelle Testinstanz der Klasse entspricht. An dieser Stelle sei nochmal zum besseren Verständnis auf die Definition von Klassen in Kapitel 3.3.1 verwiesen. Der *Score* wird mathematisch mit einer reellen Zahl in einem Wertebereich zwischen 0 und 1, die Grenzen eingeschlossen, ausgedrückt. Je näher dieser Wert bei 1 liegt desto mehr ähnelt die Testinstanz dem gegebenen Muster bzw. der

[2]Es wird hier auch von einer *Score* gesprochen.

Klasse und desto wahrscheinlicher ist es, dass der Gegner einen indirekten Schuss auf das Tor versuchen wird.
Neben den bisherigen Hauptfunktionen der Pattern-Klasse existieren noch weitere kleinere aber notwendige Methoden, die eine bessere Handhabung ermöglichen. Grundsätzlich soll jedes Muster eindeutig über einen Identifier gekennzeichnet werden. Dies ist zwar nicht zwingend notwendig erleichtert jedoch die Analyse und das Debugging der Anwendung. Mittels der Methode $initializeIndex(..)$ wird dem Muster eine eindeutige ID zugewiesen. Der Grund für diese Methode ist, dass zwar zunächst potenzielle neue Muster erkannt werden können, diese jedoch bereits anderen Mustern sehr ähnlich sind und sie deshalb nicht in die globale Liste aller Klassifikatoren aufgenommen sondern wieder verworfen werden. Die Initialisierungsmethode wird somit nur dann aufgerufen, wenn ein völlig neues Muster erkannt wurde.

4.6 Persistierung bekannter Muster

Über einen Gegner gewonnene Informationen sollten natürlich auch bei weiteren Spielen und Wettkämpfen genutzt werden können und nicht nur zur Laufzeit einer Sumatra-Instanz bekannt sein. Deshalb ist es notwendig erkannte Spielmuster in einer Datei abzulegen und diese bei einem erneuten Start von Sumatra wieder herzustellen.
Um diesen Anforderungen gerecht zu werden, kommen bestehende Java-Technologien zur Serialisierung zum Einsatz. Hierzu war es notwendig zwischen, für die Speicherung, wichtigen und unwichtigen Attributen zu unterscheiden, wobei sich herausstellte, dass nur die Positionen des passenden und schießenden Roboters benötigt werden. Alle anderen Attribute können immer wieder zur Laufzeit neu generiert bzw. aus Konfigurationsdateien gewonnen werden. Attribute die nicht mit serialisiert werden sollen werden entsprechend mit dem Schlüsselwort *transient* markiert und sind somit von den Standard-Serialisierungprozessen ausgeschlossen.
Damit die Klasse grundsätzlich *Pattern* persisitiert werden kann muss sie das Interface Serializable implementieren. Die Klasse *PatterListSerializer* hat dabei die Aufgabe eine Liste von Patterns abzuspeichern bzw. zu laden.

4.7 Verhinderung von Angriffsmustern

Da nun eine Klassifizierung der aktuellen Instanz bezogen auf potenzielle indirekte Schüsse vorliegt kann darauf aufbauend ein Spielzug entwickelt werden, der es dem Gegner erschwert den potenziellen Angriff durchzuführen.
Dieser Spielzug umfasst genau einen Roboter, der sich so positioniert, dass er das wahrscheinlichste Angriffsmuster blockiert. Dabei soll sich diese Rolle auf den Roboter konzentrieren, der den möglichen Schuss auf das Tor durchführt. Zur Berechnung der Position des eigenen Roboters wird eine Linie zwischen dem gegnerischen Robtorer und dem Mit-

telpunkt des eigenen Tors gezogen. Auf dieser Linie positioniert sich dann die Rolle mit einem Abstand von 20cm vom Gegner. In der folgenden Abbildung 4.5 ist die Zielposition der Rolle rot markiert.

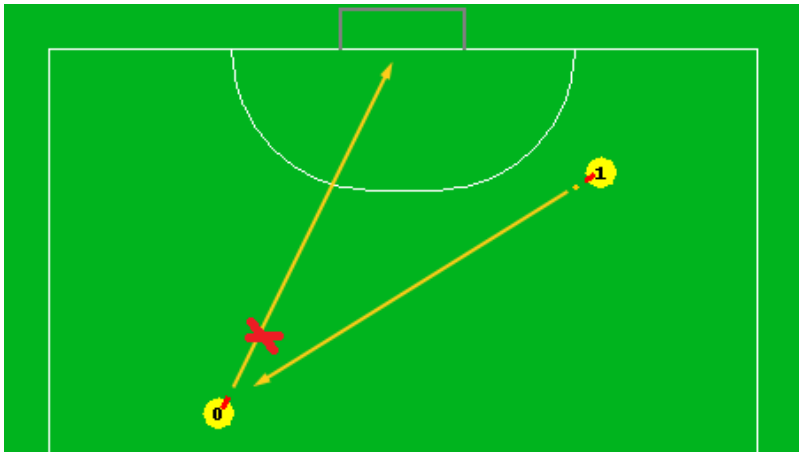

Abbildung 4.5: Position des Verteidigers.

Kapitel 5

Zusammenfassung und Ausblick

In dieser Studienarbeit wurde erfolgreich ein Verfahren geschaffen das es ermöglicht gegnerische indirekte Schüsse zu erkennen. Dabei wurde ein Framebuffer benutzt um in der Vergangenheit liegende Spielsituationen auswerten zu können und so zu bestimmen ob ein indirekter Schuss statt gefunden hat oder nicht. Weiterhin wurde umgesetzt das gleiche Muster, bei welchen Passer und Schütze an ungefähr der selben Stelle stehen wie bei bereits erkannten Mustern, nicht noch einmal erkannt werden um so die auszuwertende Datenmenge zu begrenzen. Aufgrund dieser Daten wurde ein exemplarischer Spielzug geschaffen der diese indirekten Schüsse abwehren kann, indem er den Schuss des Schützen blockt.

Literaturverzeichnis

[1] ELEKTRONIKPRAXIS: *Bremer Studenten wurden Europameister bei den RoboCup German Open 2009*. `http://www.elektronikpraxis.vogel.de/index.cfm?pid=4800&pk=185583&sector=2`, Abruf: 09. Januar 2012

[2] Ian H. Witten, Eibe F.: *Data Mining Praktisce Werkzeuge und Techniken für das maschinelle Lernen*. Hanser, 2001. – ISBN 3–446–21533–6