



BACHELORE THESIS

Die motorische Ebene einer künstlichen Intelligenz

am Beispiel der Vienna Cubes

TECHNIKUM WIEN
Fachhochschulstudiengang Elektronik

Stefan Leipold
Matr.Nr. TW03a005
Ameisgasse 63/1/20
1140 Wien

FH-Prof. Dipl.-Ing. (FH) Alexander Hofmann

Wien, 9. Juni 2006

EIDESSTATTLICHE ERKLÄRUNG

Hiermit versichere ich, die vorliegende Arbeit selbstständig und unter ausschließlicher Verwendung der angegebenen Literatur und Hilfsmittel erstellt zu haben.

Die Arbeit wurde bisher in gleicher oder ähnlicher Form keiner anderen Prüfungsbehörde vorgelegt und auch nicht veröffentlicht.

Ich erteile hiermit der Fachhochschule Technikum Wien die wissenschaftlichen Nutzungsrechte an der vorliegenden Arbeit.

Wien, 9. Juni 2006

Abstract

The artificial intelligence (AI) is one of the most important parts of every Robo-Cup team. This software framework handles the robots, selects the moves during the game and evaluates tactical strategies. The following paper describes how an AI design could look like and especially how the data communication between each module could be designed. Further more an explanation how to build and how to use basic skills in this system is included.

So the reader is able to handle the vienna-cubes artificial intelligence, to design own skills and behaviours and other additional modules.

Abstract	1
1 Glossar	1
2 Einleitung	2
2.1 Challenges	2
3 Künstliche Intelligenz	3
3.1 Datenaustausch	3
3.2 Gliederung der KI	5
3.3 Antizipatorischen Zentrum	5
3.4 Sensorisches Zentrum	7
3.5 Deliberatives Zentrum	8
3.6 Motorisches Zentrum	9
4 Schnittstellen	11
4.1 Klasse FieldObject	12
4.2 Klasse Robot	12
4.3 Klasse Ball	12
4.4 Klasse Object2D	13
4.5 Vektordaten	14
5 Behavior Programmierung	15
5.1 Datenfluss	15
5.2 Klasse: Behavior	17
5.3 Vorhandene Behaviors	18
5.4 FieldDataProxy	20
6 Skill Programmierung	21
6.1 Klasse: Skill	21
6.2 Klasse: SkillRunner	21
6.3 Ablauf eines Skills	22
6.4 Vorhandene Skills	23
7 RobotBrain und Motivator	25
7.1 Motivator	25
7.2 RobotBrain	26
7.3 Datenaustausch	26
8 Anhang A: Starten der OKI	28
9 Anhang B: Installation der OKI	29
10 Literaturverzeichnis	31

1 Glossar

Back Spin Device:	Eine Walze am Roboter zur Ballführung.
KI:	Abkürzung für „künstliche Intelligenz“.
Kicker:	Schußeinrichtung am Roboter.
Lichtschanke:	Sensor am Roboter, gibt zurück ob der Roboter im Besitz des Balles ist.
OKI:	Abkürzung für „organische, künstliche Intelligenz“.
Prediction:	Hier: Der Berechnungsvorgang von zukünftigen Positionsdaten auf Grund von vergangenen Positionsdaten.
Referee-Box:	Schnittstelle zwischen Schiedsrichter und KI bzw. OKI.
SDO:	Abkürzung für „Shared Data Object“, ein System zum Datenaustausch.

2 Einleitung

Diese Arbeit beschäftigt sich mit der Ansteuerung und Programmierung autonomer Roboter, der künstlichen Intelligenz (im Folgenden KI). Das Hauptaugenmerk ist auf die motorische Ebene der KI, d.h. auf die Steuerung und Koordination der Roboterbewegungen gelegt. Am Beispiel der KI des Roboter-Fußball-Teams Vienna Cubes soll der Aufbau dieser Ebene detailliert beschrieben werden. Aufgrund des engen Zusammenspiels der verschiedenen Ebenen der KI wird auch kurz auf den gesamten Aufbau und insbesondere auf die Kommunikation zwischen den verschiedenen Modulen eingegangen. Der Leser erhält somit einen Gesamtüberblick über die KI der Vienna Cubes womit es ihm ermöglicht wird, diese in Betrieb zu nehmen und mit ihr zu arbeiten. Weiters enthält die Arbeit detaillierte Informationen über die Entwicklung von Skills und Behaviours sowie deren Anwendung und kann somit als „HowTo“ für die Programmierung eigener Module verwendet werden.

2.1 Challenges

Die gesamte Arbeit orientiert sich in erster Linie an der Aufgabenstellung der Challenges. Diese Aufgaben sind ein Teil des RoboCups¹ bei denen in erster Linie die mechanischen Eigenschaften und die Bewegungsfähigkeit des Roboters selbst zu tragen kommen, ebenso werden Fähigkeiten wie Passen, Ballannahme und das Ausweichen vor Hindernissen getestet.[3]

¹Offizielle Website: www.robocup.org

3 Künstliche Intelligenz

Im Gegensatz zu den drei klassischen Architekturen der künstlichen Intelligenz([4]), verfolgt die KI der Vienna Cubes einen komplett neuen Ansatz. Die gängigen klassischen Architekturen besitzen einen fix festgelegten Ausführungspfad um ein Problem zu lösen und es sind jeweils nur die benötigten Programmteile aktiv. Weiters ist die Reihenfolge in der diese Programmteile abgearbeitet werden zumeist auch unumstößlich in den Sourcecode integriert. Die Daten selbst werden linear von einer Methode zur nächsten weiter gegeben (Abb.:3.1).

Bei der Entwicklung des Konzepts für die neue KI wurde das Nervensystem eines Lebewesens zum Vorbild genommen. Für die Programmierung dieser organischen künstlichen Intelligenz (OKI), auch auf den untersten Ebenen der Implementierung, bedeutet das, dass sich das Gesamtsystem aus einer Vielzahl kleinerer Module und Verarbeitungseinheiten zusammensetzt die unabhängig von einander arbeiten. Sämtliche Programmteile werden parallel ausgeführt, auch wenn die Daten die sie produzieren im Moment nicht gebraucht werden. Da alle Daten, wie in einem Nervensystem, jeder Zeit für jedes Verarbeitungszentrum zugänglich sein müssen, ist auch der Datenaustausch zwischen den Modulen ein wichtiger Punkt.

3.1 Datenaustausch

Der Datenaustausch zwischen den Verarbeitungszentren und den einzelnen Modulen stellt gleichzeitig auch das Kurzzeitgedächtnis des Systems dar und kann mit den Nervenverbindungen eines Lebewesens gleichgesetzt werden. Prinzipiell hat jedes Verarbeitungszentrum die Möglichkeit Daten aus diesem Speicher zu lesen aber auch Daten zurück zuschreiben. Der Speicherzugriff erfolgt über logische Kanäle die Fiber genannt werden. Jedes Fiber bietet Zugriff auf genau einen Datensatz. D.h. werden Daten aus der Erinnerung benötigt, muss es zu jedem dieser Datensätze auch ein Fi-

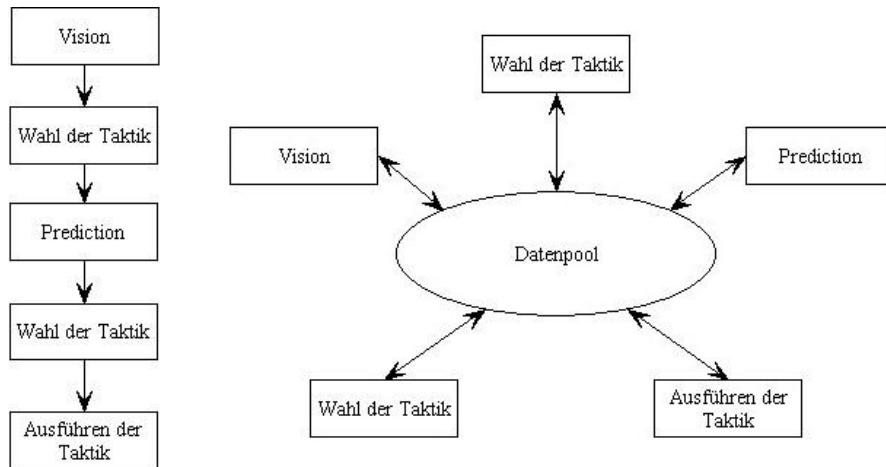


Abbildung 3.1: Unterschiedliche Datenverarbeitung zwischen KI (li.) und OKI (re.)

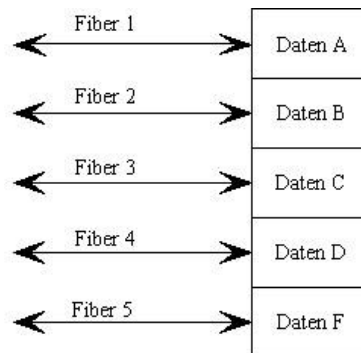


Abbildung 3.2: Beispiel eines Shared Data Object (SDO)

ber geben. Um die Übersicht zu bewahren werden mehrere Fiber zu einem Bundle zusammen geschlossen (Abb.:3.2).

Softwaretechnisch ist diese Schnittstelle als Shared Data Object (SDO) realisiert. Die benötigten Daten können über IDs abgerufen bzw. geschrieben werden. Eine Liste der verwendeten IDs befindet sich in Tabelle 4.1. Weiteres verfügt jedes SDO über eine Ethernetschnittstelle. Dadurch ist es möglich einzelne Module oder Verarbeitungszentren auf verschiedene Rechner auszulagern. Wird ein SDO beschrieben, werden die Daten automatisch an alle anderen SDOs im Netzwerk versendet. Der Umstand dass dadurch neuere Daten durch ältere Werte ersetzt werden können, wird bewusst in Kauf genommen. Eine genauer Erklärung dazu findet sich in [4]

3.2 Gliederung der KI

Die OKI kann im Groben in vier große Verarbeitungseinheiten unterteilt werden:

- Im **antizipatorischen Zentrum** erfolgt die Berechnung der zukünftigen Roboter- und Ballpositionen, um die zeitliche Verzögerung der weiteren Verarbeitungseinheiten auszugleichen.
- Das **sensorische Zentrum** beinhaltet die Ansteuerung und Auswertung der verwendeten Sensoren (Vision, Referee-Box, Funkmodul).
- Das **motorische Zentrum** ist für die Ansteuerung und das Handling aller Akteure (=Roboter) verantwortlich. Hier sind auch die, in der Einleitung erwähnten, Behaviors und Skills angesiedelt.
- Das **deliberative Zentrum** übernimmt die Auswahl und Planung von Spielzügen, Taktiken und verteilt die einzelnen Aufgaben unter den Robotern.

3.3 Antizipatorischen Zentrum

Da sämtliche Berechnungen und Datenübertragungen in der OKI, so wie in jeder anderen KI Zeit benötigen, reagiert das System auf jede Situation zeitverzögert. Die Zeitverzögerung entspricht der Durchlaufzeit vom Empfangen und Auswerten der Bilder (Vision), formulieren der Befehle (KI) bis zur Reaktion der Roboter auf die gesendeten Befehle (mechanische Trägheit). Bei einem Roboter Fußballspiel betrifft das in erster Linie die Positionsdaten der eigenen und gegnerischen Roboter sowie die des Balles. Würde man das System direkt mit den Daten der Vision speisen, würde der Roboter immer einen Zeitschritt zu langsam sein um z.B. den Ball effektiv abzufangen.

Um dem vorzubeugen, berechnet das antizipatorische Zentrum die zukünftigen Positionen der Roboter und des Balles. Für diese Berechnung kommen drei verschiedene Methoden in Verwendung, wobei die Auswahl entweder manuell (über die Konfiguration der OKI) oder dynamisch während der Spielzeit erfolgt. Die Abbildung 3.3 zeigt den groben Aufbau dieses Verarbeitungszentrums.

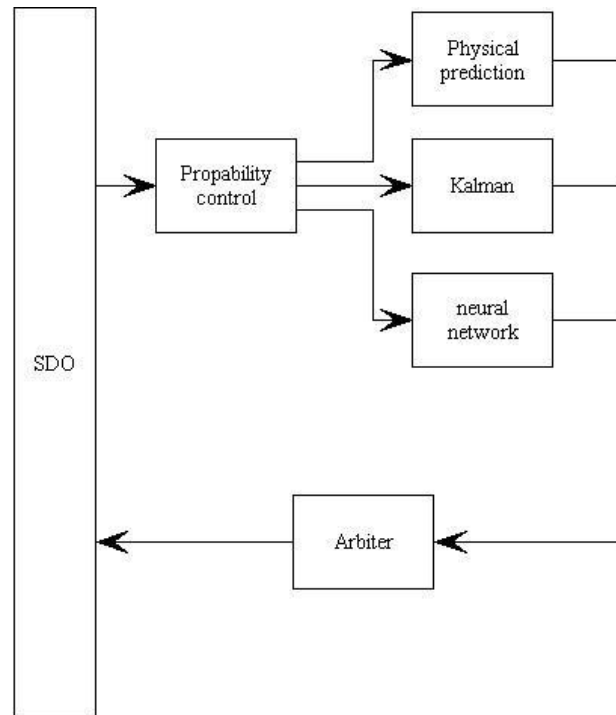


Abbildung 3.3: Aufbau des Antizipatorischen Zentrums

- **Propability control:** Hier werden die Daten von der Vision auf ihre Richtigkeit überprüft. Diese Überprüfung stützt sich in erster Linie auf physikalische Gegebenheiten, d.h. alle Objekte am Feld können sich nur mit endlicher Geschwindigkeit bewegen, dadurch kann sich ein Objekt zwischen zwei Frames nur in einem bestimmten Radius weiter bewegt haben, andernfalls sind die empfangenen Daten nicht korrekt.
- **Physikalische Berechnung** Die physikalische Berechnung baut auf einer virtuellen Nachbildung des Spielfeldes¹ und der sich darauf bewegende Objekte auf und versucht über physikalische Gesetze eine zukünftige Position zu berechnen.
- **Kalmanfilter:** Der Kalmanfilter ist der bekannteste stochastische Zustandsschätzer auf diesem Teilbereich der Robotik [5]. Mit seiner Hilfe wird zum Einen das Rauschen des Eingangssignales gedämpft und zum Anderen bietet er die Möglichkeit zukünftige Positionen zu schätzen. Ein weiterer Vorteil gegenüber ähnlichen Systemen zeigt sich in der Echtzeitfähigkeit des Filters.

¹Stützt sich auf die Methoden aus den ODE („Open Dynamics Engine“)Libraries (www.ode.org)

- **Neuronale Netze:**² Die einfachste Variante der Positionsvorrausberechnung, da keinerlei physikalischen Eigenschaften benötigt werden. Das Verhalten des Netzes muss dafür aber in einer Lernphase antrainiert werden.
- **Arbiter:** Der Arbiter vergleicht die vorausberechneten Positionen mit den später tatsächlich Erreichten und fällt somit die Entscheidung darüber, welche der drei Systeme die niedrigste Fehlerrate aufweist. Die so errechneten Daten werden dem SDO übergeben.

3.4 Sensorisches Zentrum

Das sensorische Zentrum kann mit den Sinnesorganen eines Lebewesens gleichgesetzt werden. Hier werden alle empfangenen Messwerte und Befehle aufbereitet und über das SDO in das System eingespeist. Im derzeitigen Entwicklungsstadium umfassen die Sensoren:

- **Referee-Box** empfängt die Steuerbefehle vom Schiedsrichter. Eine genaue Beschreibung der Box und der verwendeten Befehle findet sich unter [2]
- **Vision** Die Vision überträgt die aktuellen Positionen der Roboter und des Balles.
- **Roboter** Die Roboter selbst besitzen die Möglichkeit, Statusinformationen (Akku-, Kickerspannung, Lichtschranke zur Ballerkennung) an die OKI weiterzugeben.
- **GUI** Über das graphical user interface können auch Steuer- und Konfigurationsdaten an das System übermittelt werden.

Da dieses Verarbeitungszentrum auch das Interface zum Funkmodul und daher die Verbindung zu den Robotern herstellt, werden hier auch die Daten die das motorische Zentrum produziert ausgelesen und an die Aktoren (Roboter des eigenen Teams) weiter geleitet.

²Stützen sich auf die Methoden der FANN („Fast Artificial Neural Network“) Library (leenissen.dk/fann/)

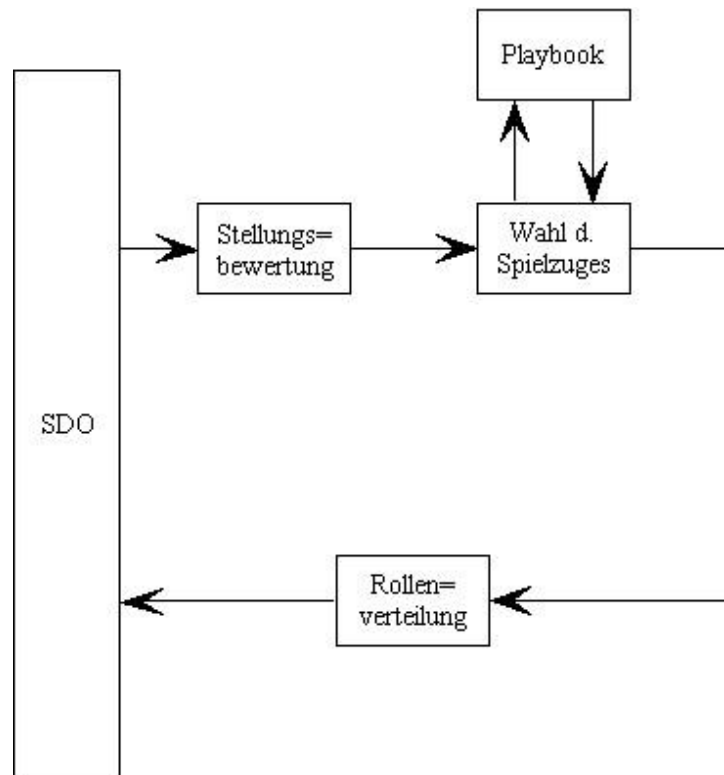


Abbildung 3.4: Aufbau des deliberativen Zentrums

3.5 Deliberatives Zentrum

Das deliberatives Zentrum übernimmt die eigentliche Denkarbeit, die den Ablauf des Spieles bestimmt. Seine Hauptaufgaben sind die Stellungsbewertung, Spielzugwahl, Rollenvergabe, Ausführung des Spielzuges und die Überwachung der gewählten Aktion. Im derzeitigen Stadium der Implementierung ist dieses Zentrum noch nicht vollständig vorhanden, da die Bewältigung der Challenges ohne dieses Modul auskommt. Die Abbildung 3.4 beschreibt daher nur den theoretischen Aufbau dieser Verarbeitungseinheit.

- **Stellungsbewertung:** In erster Instanz werden die Positionen aller Objekte auf dem Spielfeld nach ihrer Vorteilhaftigkeit bewertet. Dies dient zum Einen als Grundlage für die Wahl des nächsten Spielzuges und zum Anderen ist sie ein Teil der Spielzugüberwachung. Eine mögliche Realisierungsmethode für dieses Teilmoduls ist ein Potentialfeld, das jeder Position am Spielfeld einen Wert zuweist der, situationsabhängig, proportional zur Vorteilhaftigkeit dieser Position

ist.

- **Gamebook** Das Gamebook ist eine Liste aus vorgefertigten Spielzügen für bestimmte Situationen. Diese Schablonen sollen aber keinen exakten Ablauf festlegen, sondern nur als Vorlage dienen.
- **Wahl des Spielzuges** Dieses Teilmodul wählt, auf Grund der Stellungsbewertung, aus dem Gamebook den nächsten Spielzug aus. Während des Spielzuges überprüft diese Einheit ob der Spielzug korrekt abläuft und bricht ihn anderenfalls ab und wählt einen Vorteilhafteren aus.
- **Rollenverteilung** Die Aufgabenverteilung unter den Robotern basiert in erster Linie auf den Daten der Stellungsbewertung und den Anforderungen die der gewählte Spielzug stellt.

3.6 Motorisches Zentrum

Im motorischen Zentrum ist das gesamte Verhalten der Roboter beschrieben. Begonnen vom grundlegenden Verhalten der Bots (z.B. „Fahre von A nach B“ oder „Dreh dich um die eigene Achse“) sind hier auch komplexere Bewegungsabläufe beschrieben (z.B. „Hol den Ball und fahr damit nach A“ oder „Hol den Ball und passe zu X“). Jeder dieser Bewegungsabläufe wird überwacht und kann jeder Zeit abgebrochen werden. Abb. 3.5 verdeutlicht die Zusammenhänge der Teilmodule im motorischen Zentrum.

- **Behaviors:** Die Behaviors bilden die unterste Ebene der Bewegungsabläufe. Hier ist beschrieben, wie sich der Bot bei einer einfachen Bewegung (z.B. Drehung um die eigene Achse) verhält und welche Befehle an ihn gesendet werden müssen um diese Bewegung zu erhalten.
- **Skills:** Die Skills bauen auf den Behaviors auf und nutzen deren Bewegungsbeschreibung um komplexere Abläufe zu generieren (z.B. Ball holen und Passen).
- **Robot Brain:** Für jeden einzelnen Roboter am Spielfeld verwaltet das motorische Zentrum ein eigenes Robotbrain. Diese Teilmodule starten, beenden und

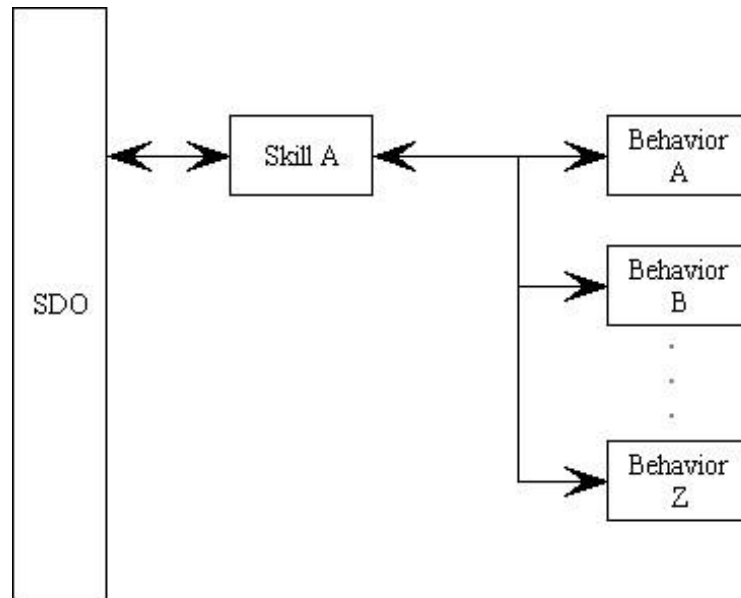


Abbildung 3.5: Grober Aufbau des motorischen Zentrums

überwachen die Skills der Roboter. Somit erscheint für das deliberative Zentrum jeder Roboter als selbstständige Einheit, die ohne äußeres Zutun komplexere Befehle ausführen kann.

4 Schnittstellen

Die OKI interne Datenübertragung erfolgt ausschließlich über, das schon beschriebene, SDO. Über die entsprechende ID erhält man Zugriff auf ein Objekt, das sämtliche Daten enthält, die von einem speziellen Teilbereich der OKI gebraucht werden. Ein solcher Oberbegriff wären zum Beispiel die Visiondaten. Die Tabelle 4.1 enthält die Auflistung aller internen Schnittstellen im Zusammenhang mit den benötigten Klassen um auf die korrekten Daten zugreifen zu können.

SDO ID	Name	Schnittstelle	Klasse
1	SDI_FO_RADIO_TEAM0	OKI → SIM/BOT/GUI	FieldObjects
2	SDI_FO_RADIO_TEAM1	OKI → SIM/BOT/GUI	FieldObjects
3	SDI_FO_VISION_TEAM0	Vision → OKI	FieldObjects
4	SDI_FO_VISION_TEAM1	Vision → OKI	FieldObjects
10	SDI_FO_KALMAN	Prediction → OKI	FieldObjects
15	SDI_FO_PHYSIKAL	Prediciton → OKI	FieldObjects
20	SDI_FO_MANUAL	GUI → OKI	FieldObjects
9991	TRANS_IDS[1]	Skills → Commandtrans.	Object2D
9992	TRANS_IDS[2]	Skills → Commandtrans.	Object2D
9993	TRANS_IDS[3]	Skills → Commandtrans.	Object2D
9994	TRANS_IDS[4]	Skills → Commandtrans.	Object2D
9995	TRANS_IDS[5]	Skills → Commandtrans.	Object2D

Abbildung 4.1: Shared data object IDs

Im Folgenden sollen nun alle Objekte die zum Datenaustausch herangezogen werden, beschrieben werden, wobei die Setter und Getter Methoden hier nicht genauer

beschrieben werden. Eine Definition der externen Schnittstellen (zwischen OKI und Roboter, OKI und Refereebox) finden sich in [2].

4.1 Klasse FieldObject

Die Klasse „FieldObjects“ enthält die Daten aller am Feld befindlichen Objekte.

Eigenschaften

Ball ball:	“ball“ vom Typ Ball speichert alle Daten betreffend den Ball
Robot ownRobots[MAX_NUM_ROBOTS]:	Array vom Typ Robot, enthält die Daten der eigenen Roboter.
Robot oppRobots[MAX_NUM_ROBOTS]:	Array vom Typ Robot, enthält die Daten der gegnerischen Roboter.

4.2 Klasse Robot

Die Klasse Robot enthält die Daten eines Roboters.

Eigenschaften

int X, Y, R	Enthält die Koordinaten des Roboters, sowie seine Drehrichtung. Dimension der Einheiten.
int dX, dY, dR	Beschreibt die Geschwindigkeit in der jeweiligen Bewegungsrichtung bzw. die Drehrichtung.
char Quality	Wird von der Vision gesetzt und gibt an wie genau die Positions- und Geschwindigkeitsdaten sind. -1 beschreibt ein ungültiges Paket.

4.3 Klasse Ball

Enthält Positions- und Bewegungsdaten des Balles.

Eigenschaften

int X, Y	Enthält die Koordinaten des Balles.
int dX,dY	Beschreibt die Geschwindigkeit in der jeweiligen Bewegungsrichtung.
char Quality	Wird von der Vision gesetzt und gibt an wie genau die Positions- und Geschwindigkeitsdaten sind. -1 beschreibt ein ungültiges Paket.

4.4 Klasse Object2D

Beschreibt das exakte Verhalten eines Roboters. Im Gegensatz zur Klasse Robot umfasst diese Beschreibung auch die Bewegung der „Back Spin Device“ und des „Kickers“, sowie einen Bounding Circle, der einen Bereich um den Roboter absteckt.

Eigenschaften

int _team	Teamnummer (0, 1)
int _id	Nummer im Team (0 - MAX_NUM_ROBOTS)
int _rotation	Winkel des Roboters
Vector2D _position	Position des Roboters
Vector2D _movement	Bewegung des Roboters
int _dribblerSpeed	Geschwindigkeit der „Back Spin“
int _kickPower	Stärke des Schusses
bool _kickWhenReady	true: Roboter schießt sobald er in Ballbesitz ist
Vector2D _boundingCircleCenter	Mittelpunkt des Boundingcircle (=Position des Roboters)
double _boundingCircleRadius	Radius des Boundingcircle.

Methoden

- **bool isPointInBoundingCircle(Vector2D point)**

Überprüft ob der angegebene Punkt innerhalb des Boundingcircle ist.

- **bool isPointInBoundingCircle(Vector2D point, double rmod)**
Überprüft ob der angegebene Punkt innerhalb des Boundingcircle ist, wobei der Radius um den Wert „rmod“ verkleinert oder vergrößert werden kann.
- **bool resetToBounds()**
Setzt den Positionsvektor auf einen Punkt innerhalb des Spielfeldes.

4.5 Vektordaten

Sämtliche Spiele werden auf einem Spielfeld, mit den Abmessungen 4,90m x 3,50m ausgetragen. Systemintern wird jedoch eine andere Maßeinheit verwendet. Der Koordinatenursprung liegt im Zentrum des Spielfeldes (Abb.:4.2). Die X-Richtung wird in +/-2450 und die Y-Richtung in +/- 1700 Einheiten aufgeteilt.

Sämtliche Winkel werden in „Centigrad“ angegeben. Als Ursprung zur Messung absoluter Winkel dient die X-Achse.

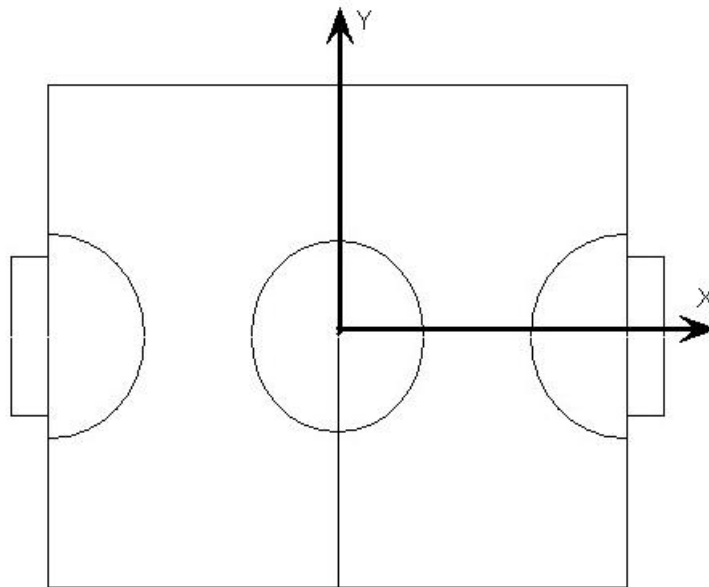


Abbildung 4.2: Schematische Darstellung des Spielfeldes

5 Behavior Programmierung

Im folgenden Kapitel soll nun detailliert beschrieben werden, wie Behaviors aufgebaut sind und wie neue Behaviors programmiert werden.

Die Behaviors beschreiben, auf welche Art und Weise der Roboter einfachste Aktionen ausführt. Die strenge Unterscheidung zwischen Behaviors und Skills bringt den Vorteil mit sich, dass, im Falle eines kompletten mechanischen Umbaus des Roboters, nur diese Klasse verändert bzw. neu erstellt werden muss, da nur sie auf das Verhalten des Roboters abgestimmt sein muss. Die darüber liegenden Skills greifen in weiterer Folge nur noch auf die Behaviors zu und sind somit vom Verhalten des Roboters unabhängig.

Sämtliche, zurzeit verwendete Behaviors sind als Methoden der Klasse „BotBehavior“ und „BhvAvoidance“, im Namespace „cubes:ai:tactics“, implementiert. Beide Klassen sind von der Superklasse „Behavior“ abgeleitet (siehe Abb.: 5.1).

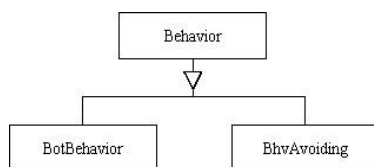


Abbildung 5.1: Ableitung der Behavior Klassen

5.1 Datenfluss

Die Behaviors selbst greifen nicht direkt auf das SDO, zu sondern erhalten alle benötigten Daten von dem Skill von dem aus sie aufgerufen werden. Dies hat zum Einen den Vorteil, dass nicht zu viele Module gleichzeitig auf das SDO zugreifen und sich somit gegenseitig blockieren. Zum Anderen, dass die Verwendungsmöglichkeit der Behaviors flexibler bleibt. Jeder Methode der „Behavior“ Klassen muss zumindest

eine Instanz der Klasse „Object2D“ übergeben werden, welche die derzeitigen Bewegungsdaten des betreffenden Roboters enthält. Nach Beendigung der Methode wird dieses Objekt wieder an den Skill zurückgegeben, wobei er die von der Behaviormethode berechneten Daten in das „Object2D“ schreibt. Somit kann ein „Object2D“ durch mehrere Behaviors gegeben werden, bevor es an die Roboter gesendet wird. Abb.: 5.2 soll dies Verdeutlichen.

Der Skill „Spiel_den_Ball_und_fahre_zur_Position“ soll aufgeführt werden. Eine In-

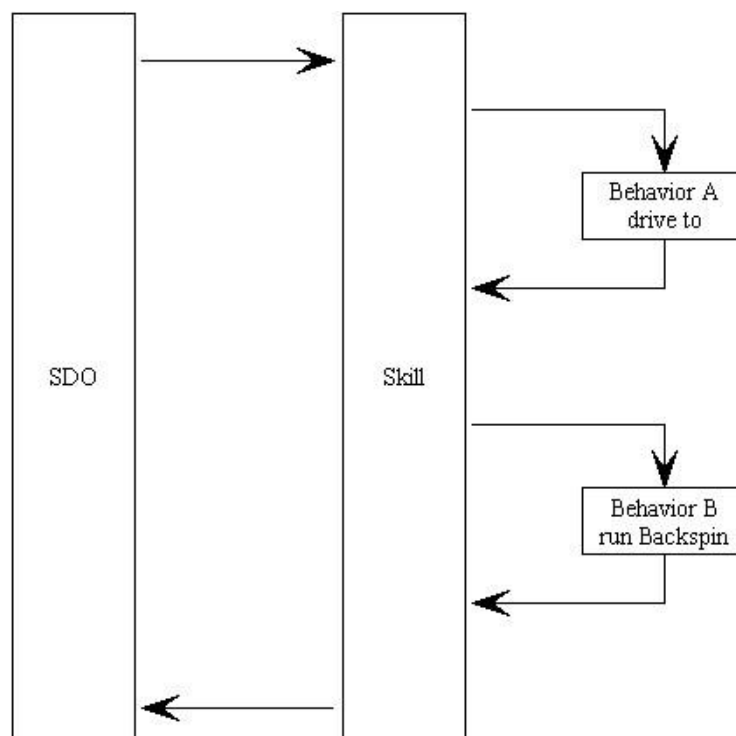


Abbildung 5.2: Datenfluss zwischen Skill und Behavior

stanz der Klasse „Object2D“ wird an Behavior A weiter gereicht. Dort werden die Bewegungsvektoren für X, Y und R berechnet um an die gewünschte Position zu kommen. Anschließend wird das „Object2D“ an den Skill zurückgegeben. Der Skill übergibt dieses Objekt nun an Behavior B. Behavior B berechnet, mit Hilfe der Fahrtvektoren wie schnell sich das „Back Spin Device“ drehen muss um den Ball nicht zu verlieren. Danach wird das „Objekt2D“ wieder an den Skill zurück gegeben.

Dieses Beispiel soll die wichtigste Regel beim Erstellen eines Behaviors verdeutlichen: Jedes Behavior darf nur die Objekteigenschaften verändern, die in seinen Ar-

beitsbereich fallen, da alle anderen Eigenschaften schon gültige Daten von vorangegangenen Behaviors sein könnten.

5.2 Klasse: Behavior

Wie schon erwähnt ist die Klasse Behavior die Superklasse aller Behaviorklassen. Sie enthält Methoden und Eigenschaften, die in allen Verhaltensschemen und für jeden Roboter vorhanden sein müssen (z.B. Festlegen der ID).

Methoden

void setBotId(int id)

Setzt die ID des Roboters der dieses Behavior verwendet.

void setTargetAccuracy(int value)

Setzt den Abstand (in mm) innerhalb dem ein Objekt am Feld als erreicht gilt.

bool isTargetReached(vg::Vector2D target, vg::Vector2D current)

Überprüft ob mit der Position „current“ die Position „target“ erreicht ist.

bool isTargetReached(vg::Vector2D target)

Überprüft ob der entsprechende Roboter die Position „target“ erreicht hat.

bool isTargetReached(vg::Vector2D target, double multiplier)

Überprüft ob der entsprechende Roboter die Position „target“ erreicht hat. Der Abstand „TargetAccuracy“ kann jedoch mit dem Multiplikator „multiplier“ verändert werden.

5.3 Vorhandene Behaviors

Im derzeitigem Status der Implementierung sind folgende Behaviors realisiert (aus Gründen der Übersichtlichkeit sind sie auf zwei Klassen aufgeteilt, dies hat aber weiter keine Bedeutung):

Klasse BotBehaviour

vg::Object2D interceptBall(vg::Object2D botPos, double maxBotSpeed, bool debug) Berechnet auf Grund der maximalen Roboter Geschwindigkeit „double maxBotSpeed“ und der Ball Bewegung einen Punkt auf dem Spielfeld, an dem der Ball abgefangen werden kann und gibt diese Position zurück. Der Parameter „debug“ gibt an ob die Bewegungsvektoren auf der GUI ausgegeben werden sollen.

vg::Object2D receiveBall(vg::Object2D ballPos, vg::Object2D botPos, double maxBotSpeed, bool debug, bool targetReached)

Berechnet einen Punkt an dem der Ball angenommen werden kann und hält ihn mit dem Backsign. Ist der Ball erreicht gibt die Methode true zurück. Der Parameter debug gibt ob die Bewegungsvektoren auf der GUI ausgegeben werden sollen

vg::Object2D driveStraightToPos(vg::Object2D start, vg::Object2D ende, bool debug) Gibt einen Bewegungsvektor zurück der von der Startposition „start“ direkt zum Ziel zeigt „ende“.

vg::Object2D driveStraightToPos(vg::Object2D target, bool isFinished, bool debug); Gibt einen Bewegungsvektor zurück der den, Roboter direkt zum Ziel bewegt. Ist das Ziel erreicht wird die Variable „isFinished“ auf „true“ gesetzt.

vg::Object2D stopAtCurrentPos(vg::Object2D target, bool debug)

Stoppt den Roboter auf der aktuellen Position.

bool getBallContact()

Gibt zurück ob der Roboter in Besitz des Balles ist oder nicht.

BhvAvoiding

vg::Object2D driveAvoiding(vg::Object2D target, bool inclBall, bool debug)

Berechnet, unter Beachtung sämtlicher andere Roboter auf dem Spielfeld, einen Bewegungsvektor um die Zielposition „target“ zuerreichen. Durch setzen des Parameters „inclBall“ wird bei der Berechnung der Bewegungsvektoren beachtet, dass der Bot im Besitz des Balles ist.

vg::Object2D driveBehindBall(vg::Object2D target, bool isFinished, bool debug)

Fährt, unter Beachtung sämtlicher auf dem Spielfeld befindlichen Roboter, eine Position hinter dem Ball an, um diesen abzufangen.

vg::Object2D kickWhenReady(vg::Object2D target, bool isFinished, bool debug)

Schießt den Ball in Richtung des angegebenen Ziels „target“. Die Methode gibt „true“ zurück wenn der Roboter in Ballbesitz ist und somit den Ball abfeuern kann.

vg::Object2D dribbleBall(vg::Object2D target, bool _BallContact, bool debug)

Aktiviert das „Back Spin Device“ und bewegt sich mit dem Ball zur Position „target“.

vg::Object2D rotateAroundBall(vg::Object2D target, bool _isFinished, bool withBackspin, bool debug)

Berechnet eine Drehung des Roboters, bei der der Ball als Mittelpunkt festgelegt ist. Der Parameter „withBacksin“ gibt an, ob während der Drehung das „Back Spin Device“ eingeschalten sein soll oder nicht.

5.4 FieldDataProxy

Da kein Behavior direkten Zugriff auf das SDO hat, aber einige Behaviors Informationen wie Ball- oder Roboterpositionen benötigen, gibt es einen zentralen Thread der in gewissen Abständen die Daten aus dem SDO liest und lokal in dem Objekt „FieldData::FieldDataProxy“ ablegt. Dieser Thread wird mit dem ersten aufgerufenen Behavior gestartet und versorgt ab diesem Zeitpunkt alle Behaviors mit den Spielfelddaten.

6 Skill Programmierung

Jeder Skill besteht aus zwei Teilen, dem Skill selbst und einer ausführenden Einheit, dem „SkillRunner“. „Der SkillRunner“ ist eine Klasse die einen beliebigen Skill beherbergen kann. Weiteres startet der „SkillRunner“ einen Thread in dem der beherbergte Skill ausgeführt wird.

Sämtlich Skills werden von der virtuellen Superklasse Skill abgeleitet.

6.1 Klasse: Skill

Die virtuelle Klasse „Skill“ legt das Grundgerüst für jeden Skill fest. Da der Skill in weiterer Folge nur in Zusammenhang mit dem „SkillRunner“ verwendet werden kann, und jeder „SkillRunner“ nur Objekte vom Typ Skill beherbergen kann, ist ein Zugriff auf den laufenden Skill nur über die Methoden aus der Superklasse Skill möglich.

Methoden

virtual void execute()=NULL

Diese Methode enthält den Code zur Ausführung des Skills und wird vom „SkillRunner“ in regelmässigen Abständen abgearbeitet.

virtual bool testTermination()=NULL

Überprüft ob gerade ein Skill läuft oder nicht.

6.2 Klasse: SkillRunner

Die Klasse „SkillRunner“ führt die benötigten Skills letztendlich aus. In einem Thread wird die Methode „execute()“ des zugewiesenen Skills in regelmässigen Abständen

ausgeführt. Die Zeit zwischen den Aufrufen der Methode „execute()“ wird dynamisch festgelegt. Somit kann ein Skill gegenüber einem Anderen zurückgestuft werden.

Methoden

void terminate()

Beendet den gerade laufenden Skill.

void operator() ()

Der Thread selbst.

void run()

Startet den Skill.

void attachSkill(Skill& skill)

Übergibt den Skill der vom „SkillRunner“ ausgeführt werden soll.

bool testTermination()

Überprüft ob gerade ein Skill abgearbeitet wird oder beendet wurde, bzw. nicht erfolgreich ausgeführt werden konnte.

6.3 Ablauf eines Skills

Grob gesehen besitzt jeder Skill den in Abb.: 6.1 beschriebenen Ablauf.

- Die **Konfiguration** befindet sich außerhalb der Methode „execute()“ des Skills. Hier werden die Befehl des RobotBrains, der dem Skill übergeordneten Instanz, festgehalten (z.B für den „MoveToPositionSkill“ die Endposition oder für den „RotateSkill“ der Endwinkel). Im Normalfall wird diese Methode für einen Zyklus einmalig aufgerufen.
- **SDO auslesen:** In diesem Schritt werden aus dem SDO die eigene Position des Roboters und alle anderen, für den Skill relevanten, Daten ausgelesen.

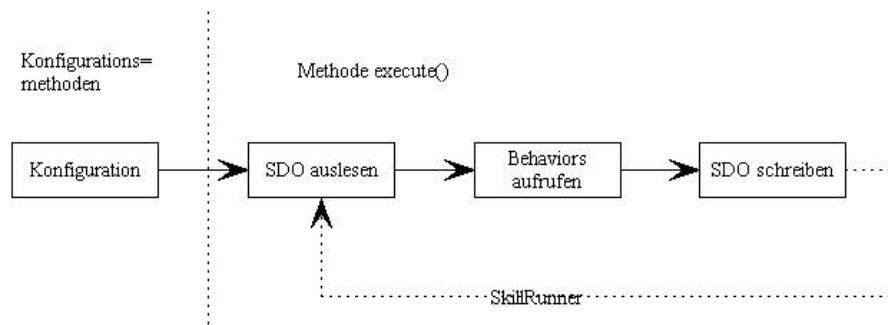


Abbildung 6.1: Ablauf eines Skills

- **Behavior-Aufruf** Nun werden die, zur Ausführung des Skills benötigten, Behaviors aufgerufen. Wie schon in Abb. 5.2 erklärt, können mehrere Behaviours hinter einander aufgerufen werden.
- **SDO schreiben** Zuletzt wird der somit berechnete Bewegungsvektor ins SDO zurück geschrieben.

Abgesehen vom ersten Punkt werden alle Schritte in der Methode „execute()“ solange ausgeführt bis der Skill erfolgreich beendet wurde, durch eine höhere Instanz abgebrochen wird oder fehlschlägt.

6.4 Vorhandene Skills

GotoBallandDribbleSkill

Roboter holt den Ball und bewegt sich mit dem Ball an eine bestimmte Position.

GotoBallandPassSkill

Roboter holt den Bal und passt zu einem zweiten Roboter

GotoPositionSkill

Bewegt den Roboter, unter Beachtung sämtlicher anderer Roboter auf dem Spielfeld, zur angegebenen Position.

MoveToAreaSkill

Bewegt den Roboter, unter Beachtung sämtlicher anderer Roboter auf dem Spielfeld, zur angegebenen Bereich.

MoveToPosSkill

Bewegt den Roboter, ohne Rücksicht auf andere Roboter, an die angegebene Position.

ReceivePassSkill

Nimmt den Ball von einem anderen Roboter an.

ShootSkill

Schießt den Ball in die angegebene Richtung.

TurnSkill

Dreht den Bot um die eigene Achse.

7 RobotBrain und Motivator

Das RobotBrain, in Verbindung mit dem „Motivator“ ist die höchste Instanz im motorischen Zentrum. Gemeinsam bilden sie die Schnittstelle zwischen den Skills und dem deliberativen Zentrum.

7.1 Motivator

Je nach Anwendung (Normales Spiel, Challenges, manuelle Steuerung) gibt es verschiedene Ausführungen des „Motivators“. Für alle Ausführungen gilt aber, dass der „Motivator“ für jeden eigenen Roboter am Spielfeld ein RobotBrain erzeugt.

Normales Spiel

Während einem normalen Spiel empfängt der „Motivator“ seine Befehle über das SDO vom deliberativem Zentrum der OKI und gibt diese an die entsprechenden RobotBrains weiter. Außerdem unterrichtet der „Motivator“ das deliberative Zentrum, über das Fehlschlagen oder den erfolgreichen Abschluss einzelner Aktionen.

Challenges

Da alle Challenges nach einem bestimmten Muster ablaufen ist, das Vorhandensein einer spielplanenden Instanz (OKI) nicht nötig. Der „Motivator“ übernimmt die Ausführung der Challenges selbständig, wobei es für jede Challenge einen eigenen „Motivator“ gibt. Einzig über die GUI können Steuerbefehle wie „Start“, „Stopp“ und „Reset“ an den „Motivator“ gesendet werden.

manuelle Steuerung

Diese Steuerung ist in erster Linie zum Testen der Roboter gedacht. Der Motivator bekommt seine Befehle entweder über die GUI (z.B. Steuern der Roboter per Maus) oder direkt vom Joystick.

Der Motivator selbst ist als Thread realisiert, der in festgelegten Abständen neue Befehle einliest und diese auf die RobotBrains aufteilt.

7.2 RobotBrain

Jedes „RobotBrain“ enthält, eine Instanz von jedem benötigten Skill und einen „SkillRunner“. Je nach erhaltenem Befehl wird der entsprechende Skill konfiguriert und in den SkillRunner geladen. Des Weiteren wird die Laufzeit mit der der „SkillRunner-Thread“ läuft angepasst. Auf diese Weise ist es möglich, einen reibungslosen Ablauf des Spiels zu gewährleisten, ohne während der Laufzeit neue Objekte (Skills) erzeugen zu müssen. Über das „RobotBrain“ wird der „Motivator“ über den Ablauf des Skills unterrichtet.

7.3 Datenaustausch

Abb. 7.1 zeigt den Aufbau und das Zusammenwirken von Motivator, RobotBrain und den Skills. Egal für welchen Modus der „Motivator“ betrieben wird, er erhält seine Steuerbefehle direkt vom SDO. Die Übermittlung der Befehle an das „RobotBrain“ erfolgt über eine ID mit der festgelegt wird welcher Skill als nächstes geladen wird und einem „Object2D“ das, je nach verwendetem Skill, die entsprechenden Steuerdaten enthält.

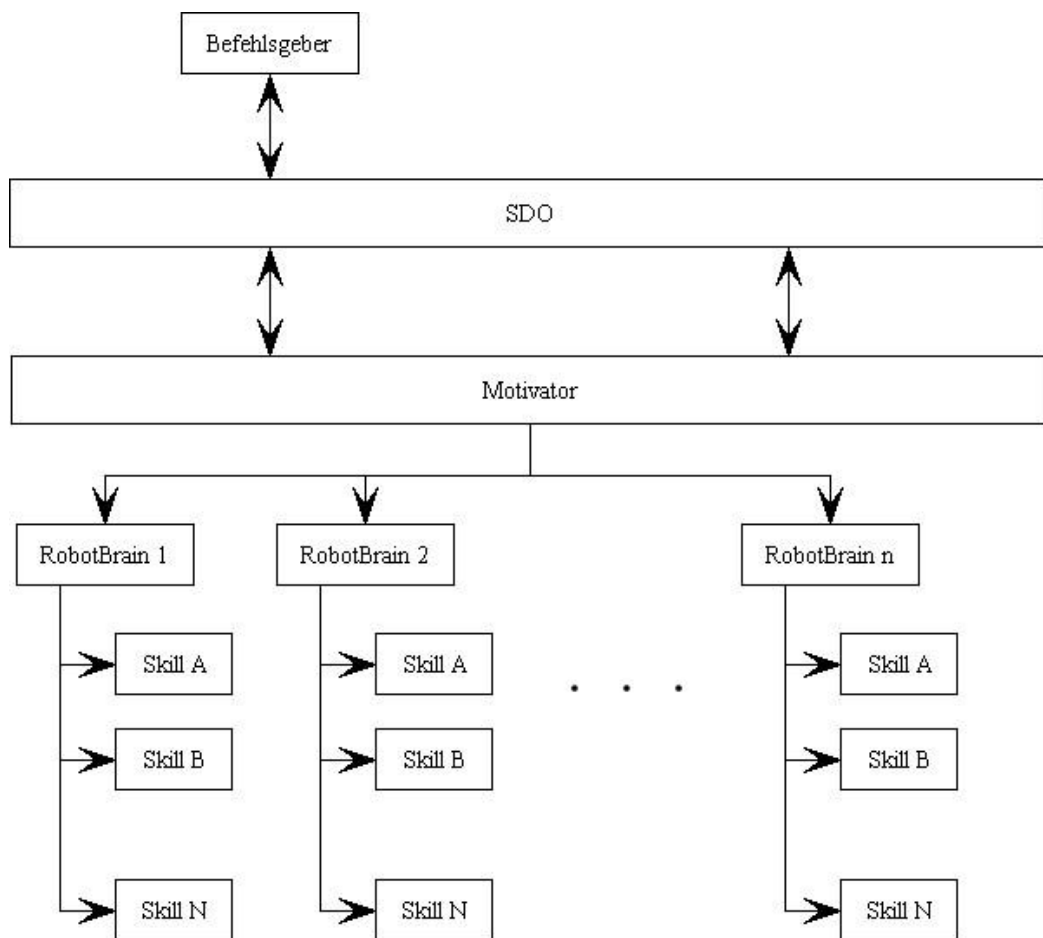


Abbildung 7.1: Zusammenspiel von Motivator, RobotBrain und Skill

8 Anhang A: Starten der OKI

Aufgrund der Modularität der OKI und da im derzeitigen Stadium der Entwicklung noch kein automatisierter Ladevorgang für die Verarbeitungszentren vorhanden ist, muss jedes Modul manuell geladen werden. Die zurzeit verwendeten Module sind:

- **SIM:** Der Pysiksimulator des Systems. Er wird einerseits dazu verwendet, um aus den aktuellen die zukünftigen Koordinaten der Roboter und des Balls zu berechnen. Auf der anderen Seite kann mit, seiner Hilfe, die Vision simuliert werden. Dadurch ist es, bis zu einem gewissen Grad, möglich, das Framework auch ohne Spielfeld und Roboter weiter zu entwickeln bzw. zu testen.
- **GUI** Das Graphical User Interface dient zur Überprüfung der Vision und/oder SIM Daten. Außerdem können von hieraus manuelle Steuerbefehle an die OKI gesendet werden.
- **Command Transmitter:** Der „Command Transmitter“ sendet die endgültigen Steuerdaten über das Funkinterface an die Roboter, bzw., im Testmodus, zurück an den SIM.
- **Motivator:** Eine genaue Beschreibung des „Motivators“ findet sich im Kapitel RobotBrain.
- **DSMListener:** Der „data shared memory listener“ legt den gemeinsamen Speicherplatz für das SDO an und überwacht diesen.

Um die Funktionsfähigkeit des Systems zu gewährleisten, müssen diese Module in der Reihenfolge gestartet werden, in der sie oben aufgelistet sind.

9 Anhang B: Installation der OKI

Zur Entwicklung der OKI wird zur Zeit „**Visual Studio .NET**“ verwendet, da sämtliche zusätzlich verwendete Libraries (z.B. DirectX, FANN, ODE) von dieser Entwicklungsumgebung unterstützen werden. In dieser Installationsanweisung wird nicht weiter auf die genaue Konfiguration von Visual Studio eingegangen, da diese Einstellungen rechnerbedingt abweichen können.

Source Code

Der Source Code kann zur Zeit über den SVN Server der Vienna-Cubes heruntergeladen werden. Hierbei ist zubeachten, dass die, dort angegebene, Verzeichnisstruktur nicht verändert werden darf. Lediglich die Trunk-Verzeichnisse werden bei der lokalen Kopie nicht verwendet. D.h. der Source Code aus dem Verzeichnis „`../ai/tactics/trunk/`“ am SVN Server wird lokal im Verzeichnis „`../ai/tactics/`“ abgespeichert.

Weitere Programme und Libraries

Bevor die einzelnen Module der OKI kompiliert werden können, müssen noch zusätzliche Libraries und Programme installiert werden. Die Programmbezeichnungen, die hier gegeben werden, entsprechen dem aktuellen Stand der Entwicklung, können aber später durch neuere Versionen ersetzt werden. Eine genaue Beschreibung der Installation für die unten genannten Komponenten befindet sich bei den Komponenten selbst.

- **Intel C++ Compiler 9.1**¹ Optimiert den Programmcode und steigert die Performance der Anwendungen.
- **DirectX 9.0c for developers**² wird vom Physiksimulator zur Darstellung der Objekte verwendet.
- **C++ Boost libraries 1.33.1**³ Stellt zusätzlich benötigte Klassen wie die Threads, SharedPointer zur Verfügung.

Kompilieren

Aufgrund des modularen Aufbaus der OKI ist es nicht nötig, ständig das gesamte Framework zu kompilieren. Wurde die gesamte OKI einmal kompiliert, ist es ausreichend, wenn immer nur die veränderten Module neu erstellt werden.

Für jedes Projekt in dem Framework sollten folgende Einstellungen für das Kompilieren verwendet werden:

- **Intel C++ Projekt** der erstellte Code sollte mit dem Intel C++ Kompiler kompiliert werden.
- **Runtime Library** auf **Multi-threaded Debug (/MTd)** stellen.
- keine **vorkompilierten Headerdateien** verwenden.
- **Optimization Disabled** sollte erst wenn benötigt verwendet werden.

¹www.intel.com

²www.microsoft.com/directx/

³www.boost.org

10 Literaturverzeichnis

- [1] Robo Cup Regelwerk
„<http://www.cs.cmu.edu/~brettb/robocup/rules>“, 2006
- [2] Beschreibung der Referee Box
„<http://www.cs.cmu.edu/~brettb/robocup/referee.html>“, 2006
- [3] Beschreibung der Challenges
„http://www.aichi-pu.ac.jp/ist/lab/narulab/challenge/SSL_challenge.html“,
2006
- [4] Diplomarbeit Ditmar Schreiner
„Organische Künstliche Intelligenz“, FH Technikum-Wien, 2005
- [5] Greg Welch and Gary Bishop
„An Introduction to the Kalman Filter“, University of North Carolina at Chapel Hill, 2004