HOCHSCHULE OSNABRÜCK

UNIVERSITY OF APPLIED SCIENCES

Hochschule Osnabrück
Fachbereich Medieninformatik (I&I)
Software Engineering Projekt
Leitung: Prof. Dr-Ing. Ralf Tönjes
Sommersemester 2017
Abgabetermin: 25.09.2017

Dokumentation KickIT

Projektteilnehmer:
Lukas Völler (620445), Email: voeller.lukas@gmail.com
Philip Baumgartner (568106), Email: philip.baumgartner1988@gmail.com
Viktor Koschmann (373731), Email: koschmann.viktor@gmail.com

Inhaltsverzeichnis

1 Projektbeschreibung

Bei diesem Projekt handelt es sich um ein Teilprojekt, welches die Entwicklung eines Frameworks für einen autonomen Tischfußball-Roboter als Ziel hat. Dieses Framework bietet austauschbare Algorithmen und Funktionen um es stetig erweitern zu können. Die Steuerung des Kickers erfolgt in C++ . An der Entwicklung des Frameworks beteiligten sich drei Studenten im Rahmen des Software Engineering Projekts im Sommersemester 2017. Die Aufsicht führte Prof. Dr.- Ing. Ralf Tönjes mit den beiden wissenschaftlichen Mitarbeitern Daniel Hölker (M. Sc.) und Daniel Brettschneider (M. Sc.). Die Installation des zweiten Motors übernahm Benjamin Colson, ein Austauschstudent aus Frankreich.

Als Grundlage diente ein vorbereiteter Tischkicker mit bereits installierten Komponenten. Durch einen an der Torwartstange angebrachten Linearmotor, einer über dem Tisch montierten Kamera und einem Programm zur Steuerung, ist der Computer in der Lage den Ball zu erkennen und den Torwart passend zu positionieren. Diese Erarbeitung erfolgte durch eine frühere Projektgruppe.

Somit wurden die Voraussetzungen geboten, durch eine Echtzeitanalyse der Bilddaten, die Position des Spielballs auszuwerten und den Torwart zu steuern. Diese Steuerung wurde in das Framework eingepflegt. Um eine Abwehrreihe ins Spiel zu bringen, musste ein weiterer Motor von LinMot installiert werden. Die Abwehrreihe unterstützt den Torwart bei der Ballabwehr. Die Reihe der Abwehrspieler kann neben translatorischen Bewegungen auch rotatorische Bewegungen ausführen und den Ball dadurch nach vorne schießen. Für die weitere Entwicklung des Projektes, ermöglicht das Framework eine einfache Implementierungen weiterer Spielstangen. Das langfristige Ziel dieser Projektreihe ist die Entwicklung eines kompletten Tischfußball-Roboters.

Die vorliegende Dokumentation bietet eine Beschreibung des bisher gebauten Tischfußball-Roboters und einen Überblick über die Entwicklung des Frameworks. Außerdem werden in diesem Bericht vorhandene Probleme und Herausforderungen angesprochen und Lösungsansätze vorgestellt.

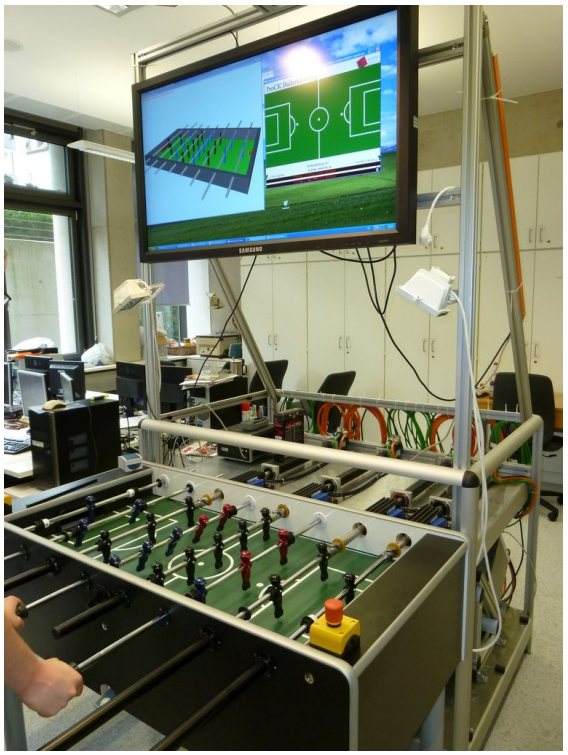
Im Folgenden wird die Projektarbeit im Hinblick auf die Technik, die Hardware und die Software beschrieben. Herausforderungen und Lösungsansätze, die sich daraus ergeben, werden nachfolgend aufgeführt. Des Weiteren werden wichtige Informationen zur Sicherheit und zur Installation bereitgestellt. Zum Schluss folgt im Fazit eine Zusammenfassung der Projektarbeit und ein Ausblick auf die Weiterentwicklung und den Ausbau des Kickertisches.

2 Stand der Technik

2.1 ProCK

Das Projekt KickIT ist nicht das Erste seiner Art. Seit 2008 befindet sich ein ähnliches Projekt namens ProCK an der Hochschule für angewandte Wissenschaften der FH München in Entwicklung. ProCK, kurz für Projekt Computerkicker, ist ein motorisierter Kickertisch, der über verschiedene Schwierigkeitsstufen verfügt. Seit 2012 ist dieser voll funktionsfähig und wird seitdem weiterentwickelt. Dies übernimmt in diesem Semester (Sommersemester 2017) eine fünfköpfige Gruppe. Grundlegende Herausforderungen wurden bereits gemeistert, deshalb kümmern sich die fünf Studenten zurzeit um die KI bzw. Spielsteuerung, die Ballerkennung und um die Fernwartung der Anlage (Hochschule München, 2017).

Die Ballerkennung im KickIT Projekt unterscheidet sich von der, die im ProCK Projekt zum Einsatz gekommen ist. Der Ball wird bei ProCK von zwei Kameras erfasst, die in gegenüberliegenden Ecken montiert sind. Sie liefern 100 Schwarz-Weiß-Bilder pro Sekunde und ermitteln die Position des Balls mittels Triangulation. Die Kameras sind nur knapp über dem Spielfeld angebracht, was den Vorteil hat, dass Spieler sich über das Spielfeld lehnen können, ohne das Kamerabild zu beeinflussen (Seck, 2011).



Die Steuerung beim ProCK-Projekt erfolgt über zwei Motoren pro Stange, einen für die Translation einen für die Rotation. Für die Bewegung der Figuren wird ein Servomotor der Firma Beckhoff genutzt. Die Rotationsbewegungen steuert ein Linearmotor der Firma Copley Controls.

Die Spielstrategie ist relativ simpel gehalten. Es wird geschossen wenn es möglich ist. Ansonsten wird darauf geachtet den Ball nur zu blockieren.

[GRAFIK]
Abbildung 1: “ProCK”
(Quelle: Hochschule München, 2011)

2.2 Kiro und StarKick

Kiro ist ein Tischkicker Roboter Projekt der Albert-Ludwigs-Universität Freiburg. Dieses Projekt resultierte in eine kommerzielle Version namens StarKick. Im Zuge dieses Projekts entstand eine Software Simulation, die es ermöglicht, Tischkicker-Roboter miteinander zu vergleichen. Bereits 2002 gab es eine fertige Version und es folgte eine Lizenzierung durch die Gauselmann AG. Bevor die Gauselmann AG mitwirkte, wurden die Stangen durch außen am Tisch angebrachte Motoren gesteuert. Auch hier findet ein Zusammenspiel zweier Motoren für Translation und Rotation statt. Nach der Lizenzierung wurde der Bewegungsmotor durch einen Seilzug unter dem Tisch ersetzt. Der Rotationsmotor befindet sich nach wie vor an der Seite des Tisches, jedoch ist er nun nicht mehr sichtbar. Das Kiro Projekt hat für ein konstantes Kamerabild sekündlich eine Kalibrierung durchgeführt. Hierbei wird der Mittelpunkt des Spielfeldes in der Bildmitte positioniert. Anschließend wird das Bild so gedreht, dass die Mittelfeldlinie 90 Grad auf dem unteren Bildrand steht. Die Bildverarbeitung wird von CMVision übernommen. Bei StarKick wird für ein gleichbleibendes Kamerabild ein anderer Weg eingeschlagen. Die Kamera wurde vom Kickertisch losgelöst und kann so beim Spielen nicht verwackelt oder versetzt werden.

Die Kamera im Prototypen Kiro war sehr anfällig gegenüber sich ändernder Lichtverhältnisse. Aus diesem Grund wurde sie in der nachfolgenden Version durch eine Infrarotkamera unter dem Tisch ersetzt. Eine weitere Ergänzung waren Infrarot LED's die ringsherum am Spielfeldrand in Ballhöhe eingelassen wurden und den Ball während jeder Bildaufzeichnung beleuchten. Die Kamera erkennt den Ball, da das Spielfeld für Infrarotlicht durchlässig ist. So ist das System zwar unempfindlich gegen Licht von außen, jedoch wird so nur noch der Ball erkannt und nicht mehr die Positionen der Spielfiguren. Die Position der Figuren kann jedoch über die Motorausrichtung ermittelt werden.

Die Position des Balls wird mit einem Differenzbild ermittelt. In der vorherigen Version des Tisches wurden wie bei KickIT Farbschwellwerte bestimmt und es konnte nur mit einem gelben Ball gespielt werden.

Die Gauselmann AG erhielt 2004 den Innovationspreis der SPD und vertreibt den ca. 350 Kilogramm schweren Kicker seit 2005 für rund 20.000€ (Schnattinger, 2006).

[GRAFIK]
Abbildung 2: "StarKick"
(Quelle: fudder, 2006)

3 Hardware

Der Kickertisch ist von der Marke Leonhart. Die zentral über dem Tisch montierte Kamera befindet sich in einer Höhe von 209,5 cm über dem Spielfeld. Die für die Steuerung der Stangen zuständigen Motoren sind auf einem Beistelltisch angebracht. Dieser ist mit dem Tisch fest verschraubt. Unter ihm befinden sich zur Feinjustage vier Plastikscheiben, die den Höhenunterschied zum Kicker ausgleichen. Die Treiber für die Kommunikation mit den Motoren sind auf einem separaten Brett verbaut. Es folgt eine Auflistung der technischen Daten der verwendeten Hardware. Der Motor für den Torwart ist der *PS01* Hub-Motor. Er ist ein Linearmotor und kann keine Drehbewegung ausführen. Er besitzt ein eigenen Treiber (C1100) sowie ein eigenes Netzteil von der Firma LinMot. Der zweite Motor ist der *PR-01* Hub-Dreh-Motor. Auch er ist von der Firma LinMot, jedoch benötigt er zwei Treiber. Einen für die Translation und einen für die Rotation.

3.1 Kickertisch

Außenmaße:	143 x 74 cm
Spielfeldmaße:	111,3 x 88 cm
Wandstärke:	31,5 mm
Höhe:	92 cm
Gewicht:	ca. 80 kg

3.2 Kamera

Name:	Basler acA640-300gc
Bildrate:	376 fps
Auflösung:	640px x 480px (VGA)
Schnittstelle:	Ethernet

3.3 Torwart Motor

Name:	Linmot PS01 Hub-Motor
Max. Kraft:	138 N
Max. Geschwindigkeit:	5,2 m/s (18,72 km/h)

3.4 Abwehr Motor

Name:	Linmot RS01 Hub-Dreh-Motor
Max. Hubkraft:	572 N
Max. Drehmoment:	8.9 Nm
Max. Geschwindigkeit Hub:	3 m/s (10,8 km/h)
Max. Rotation:	1000 rpm

4 Software

4.1 Werkzeuge

Betriebssystem:	Linux Distribution Ubuntu (16.04)
Entwicklungsumgebung:	Eclipse Neon für C++ (4.6.0)
Versionsmanagement:	Git (2.7.4)
Dokumentation:	DoxyGen (1.8.11)

4.2 Pylon

Die PylonViewer-Api wird für die Steuerung der Basler-Kamera benötigt. Sie bietet alle nötigen Treiber für die Kommunikation und Initialisierung der Kamera. Verwendet wurde die Version 5.0.1.6388-x86_64. Zu Projektstart mussten Einstellungen, wie zum Beispiel Höhe und Breite des Bildes, bei jedem Start des Computers erneut eingestellt werden. Im Verlauf des Projektes wurde das Softwarepaket in das Framework integriert. Dadurch wurde erreicht, dass alle wichtigen Einstellungen beim Programmstart automatisch vorgenommen werden.

4.3 OpenCV

OpenCV (Open Source Computer Vision Library) basiert auf C/C++ und gilt als sehr leistungsfähig. Sie wird für die Verarbeitung bzw. Auswertung der Bilder genutzt. Die Software steht für kommerzielle, sowie für nicht kommerzielle Projekte kostenlos zur Verfügung. Genutzt wurde die Version 3.3.0 für die Erkennung des Balls. OpenCV bietet eine Vielzahl an Algorithmen für Bildverarbeitung und maschinelles Sehen.

4.4 QT

QT ist ein GUI-Toolkit zur Programmierung grafischer Benutzeroberflächen. Die Bibliothek wurde für die Entwicklung des virtuellen Kickers verwendet. Zum Einsatz kam die Version qt5-default.

5 Anforderungsanalyse

Das übergeordnete Ziel dieser Projektarbeit ist es, ein Framework für die maschinelle Steuerung eines Kickertisches zu erstellen. Die Position der Spieler wird durch axiale Verschiebung ihrer Stangen so beeinflusst, dass gespielte Bälle aufhalten und ggf. zurück geschossen werden.

Es bestehen folgende Anforderungen an das Projekt. Die Umstrukturierung zu einem Framework soll die Weiterentwicklung des Tisches durch zukünftige Projektgruppen ermöglichen. Hierbei sollen zum Beispiel zusätzliche Spielfiguren bedient, oder neue Spielstrategien hinzugefügt werden können. Die Implementation neuer Bildverarbeitungsalgorithmen soll ebenfalls ermöglicht werden. Der Torwart und die Abwehr müssen den Ball halten. Das bedeutet, die Spielfiguren müssen sich bei jedem Ball, der vom Spieler geschossen wird, rechtzeitig an der Position befinden, welche den Ball daran hindert, ins Tor zu gelangen. Mittels einer Schnittstelle müssen die Bilder von der Kamera an den PC übertragen, und dort von einer Software verarbeitet werden. Anschließend wird die Ballbewegung errechnet. Aus den ermittelten Daten werden die benötigten Positionen der Stangen bestimmt und an den Treiber weitergeleitet, der den Motor und damit die Spielfiguren steuert. Das Framework muss erweiterbar, sowie verständlich und wiederverwendbar sein.

Die Stakeholder des Projekts sind: Prof. Dr-Ing. Ralf Tönjes (Leitung), Daniel Hölker und Daniel Brettschneider (Wissenschaftliche Mitarbeiter) sowie Lukas Völler, Philip Baumgartner und Viktor Koschmann als Entwickler.

Die Installation der Hardware erfolgte in Zusammenarbeit mit einem Austauschstudenten (Benjamin Colson). Im Status Quo ist das Design der Software hauptsächlich darauf ausgelegt zu funktionieren. Beim Entwurf von KickIT wird, wie oben erwähnt, eine Ausrichtung der Software auf Erweiterbarkeit, Wiederverwendbarkeit und Wartbarkeit realisiert. Ein wichtiger Teil der Umstrukturierung, ist die Untergliederung der Software in Module. Des Weiteren zählt die Definition von Schnittstellen zu einer der Hauptaufgaben.

Zu beginn der Projektarbeit erforderte das Programm bei jedem Neustart einige Benutzereingaben. Diese Prozesse, sollen so weit wie möglich, automatisiert werden. Optional kann ein User Interface für die Eingaben implementiert werden.

Im Sinne des Frameworks ist es erforderlich, für die Software eine Struktur zu definieren, die die Austauschbarkeit von Algorithmen und Funktionen und die Erweiterbarkeit der Software gewährleistet.

5.1 Musskriterien

Funktional

1. Schnittstelle zur Steuerung aller Stangen
2. Schnittstelle zur Echtzeitanalyse der Bilddaten
3. Das System muss in C++ entwickelt werden und unter Ubuntu 16.04 laufen
4. Das System muss mit der bereitgestellten Hardware kompatibel sein
5. Untergliederung der Software in Module
6. Die Erweiterbarkeit der Software, ausgerichtet auf:
    a. Das System wird in der Lage sein vier Stangen zu steuern
    b. Das System wird in der Lage sein mit den Reihen zu schießen
7. Definition von Schnittstellen zur:
    a. Austauschbarkeit der Kamera und der Motoren
    b. Austauschbarkeit von Algorithmen
    c. Austauschbarkeit von API's

Nicht Funktional

1. Das System muss erweiterbar durch zukünftige Projektgruppen sein
2. Das System muss einfach zu warten sein
3. Die Dokumentation des Codes muss verständlich sein

5.2 Wunschkriterien

Funktional

1. Eine grafische Oberfläche
2. Automatische Anpassung der Ballerkennung an die gegebenen Lichtverhältnisse
3. Anzeige von Live-Daten auf einem Display wie Schussgeschwindigkeit oder Live-Bild mit Ball-Tracking
4. Automatisierte Kalibrierung und Initialisierung der Hardware Komponenten
5. Verbesserung der Ballerkennung

5.3 Abgrenzungskriterien

"Intelligente Spielzüge" wie Passen, Stoppbälle, Push-Shots, das Verfolgen von Strategien, adaptive KI und Ballführung sind nicht zu implementieren.

6 Basler Pylon Camera Software Suite

Mit der *Pylon Software Suite*, können Basler Kameras mit allen dazugehörigen Funktionen in Betrieb genommen werden. Kamerahersteller Basler, stellt mit der Software Suite als erster überhaupt eine Software zur Verfügung, die die neue *GenICam 3-Technologie* verwendet. Dadurch ist es möglich, Kameras etwa viermal so schnell zu öffnen, wie mit GenICam-Vorläuferversionen. An Arbeitsspeicher wird ebenfalls nur in etwa die Hälfte benötigt. *GenICam* ist ein Standard der *European Machine Vision Association*, welcher über eine generische Programmierschnittstelle verfügt. Er wird zur Kontrolle von *Machine Vision Cameras* eingesetzt. Die *PylonViewer* Software bietet alle nötigen Treiber für verschiedene Kamera-Schnittstellen, die für Basler Kameras benötigt werden. Es ist möglich, mit nur wenigen Codezeilen, eigene Kamera-Anwendungen für z. B. Linux zu entwickeln. Die Kamera-Schnittstellen, die die Pylon-Software anbietet, sind z. B. *USB3 Vision*, *GigE-Vision*, *IEEE1394* und *Camera Link*. In unserem Projekt wird die *GigE-Vision* Schnittstelle verwendet. Der Pylon *GigE-Vision* Performance-Treiber trennt schnell die eingehenden Pakete, die durch die Übertragung der Bilddaten entstehen, vom anderen Datenverkehr im Netzwerk. Somit werden die übertragenen Bilddaten direkt zur Verfügung gestellt. Gleichzeitig beansprucht der Transfer wenig CPU-Ressourcen. Die Netzwerkkarte, die in unserem Projekt verwendet wird, ist eine Ein-GBit-Netzwerkkarte mit einem Intel-Chipsatz die in der Lage ist, eine hohe Übertragungsrate zu garantieren. Ein GigE-Kabel der Kategorie *Cat-6* oder höher, wird für die *GigE-Vision* Schnittstelle empfohlen.

6.1 Pylon Viewer Anwendung

In der Pylon Viewer App ist die Kamera unter Devices/GigE zu finden. Die App bietet verschiedene Einstellungen sowie Funktionen für die Kamera. Ein wichtiger Parameter für unser Projekt ist die *Packet Size*. Sie ist unter Basler(acA640-300gc)/Transport Layer zu finden. Sie wird in Byte angegeben und ist für den ausgewählten Stream-Kanal zuständig. Die standardmäßige Einstellung ist auf 9000 Byte eingestellt. Dieser Wert muss aufgrund der Netzwerkkarte des PCs auf 1500 Byte verringert werden. Für die korrekte Einstellung der Schwellwerte, welche beim Spielstart erfolgen muss, ist es wichtig, einen passenden Belichtungszeit-Parameter (ExposureTime) zu wählen. Als Hilfe kann man die automatische Einstellung der Belichtungszeit nutzen. Sie ist unter *Automatic Image Adjustment* zu finden und funktioniert nur zusammen mit der Videofunktion. Es ist ungefähr ein Drittel des automatisch eingestellten Wertes für das Spiel zu wählen. Vor dem Beenden der Pylon Viewer App, muss die Verbindung mit der Kamera getrennt werden, da die Kamera nur eine Verbindung zur gleichen Zeit bedienen kann. Dieser Zugriff ist entweder mit der App oder mit der KickIT Software möglich. In diesem Projekt wird die *CameraConfig.txt* Datei verwendet, um einige Einstellungen für die Kamera beim Spielstart automatisch vorzunehmen.

7 Architektur

7.1 Komponenten

7.1.1 BallTracking

Die *BallTracking* Einheit, zur Verfolgung des Balls, besteht aus zwei wesentlichen Komponenten. Zum Einen aus dem *BallTracker* an sich, der für die Erkennung des Balls zuständig ist. Zum Anderen aus der Kamera, die das Bild liefert. Der *BallTracker* benötigt zur Erkennung des Balles eine Kamera, so wird es durch sein Interface vorgegeben. Bei der Erstellung eines *Balltrackers* wird dann die Kamera initialisiert und er beginnt den Ball zu verfolgen. Nach dem Instanziieren und Öffnen der Kamera, werden zunächst mit *setCameraSettings()* alle nötigen Einstellungen vorgenommen, bevor sie anfängt, ein Bild zu liefern.

7.1.1.1 BallTracker

Der *BallTracker* durchläuft in seiner Funktion *startTracking()* eine Schleife, in welcher er immer wieder ein Bild von seinem *Camera* Objekt abfragt. Dieses Bild wird zunächst an die *cv::inRange* übergeben. Sie gibt ein Bild (imgThresholded) zurück. cv:: steht für den Namespace der opencv Funktionen.

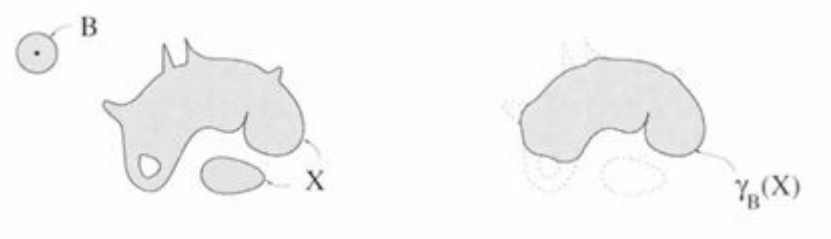
```
inRange(*cv_img,
    Scalar(threshold->blueLow, threshold->greenLow, threshold->redLow),
    Scalar(threshold->blueHigh, threshold->greenHigh,
             threshold->redHigh), imgThresholded);
```

Im zurückgegebenen Bild sind alle Pixel schwarz, die im Bereich der RGB-Werte nicht enthalten sind. Der niedere Schwellwert wird vom ersten Scalar angegeben und der höhere Wert vom zweiten Scalar. Die Werte werden in der *Camera.cpp* Datei mit der *threshold()* Funktion eingestellt und in der *CameraConfig.txt* abgelegt, von wo aus sie für die *inRange()* Funktion verwendet werden. Mit den Funktionen *cv::erode* und *cv::delate*, werden kleinere weiße Stellen aus den Bildern herausgefiltert, die kein kreisförmiges strukturierendes Element vom Durchmesser 15 aufnehmen können.

```
erode(imgThresholded, imgThresholded,
    getStructuringElement(MORPH_ELLIPSE, Size(15,15)));
dilate(imgThresholded, imgThresholded,
    getStructuringElement(MORPH_ELLIPSE, Size(15, 15)));
```

Durch das Erodieren des Bildes, wird zunächst jeder Pixel aus dem Bild herausgefiltert, dessen unmittelbare Umgebung nicht aus einer kreisförmigen Ansammlung von weiteren weißen Pixeln besteht, welche mindestens den Durchmesser 15 hat. Anschließend werden bei der Dilation des Bildes, die geschrumpften Objekte wieder auf ihre vorherige Größe gebracht.

Die folgende Abbildung zeigt ein Beispiel mit dem strukturierenden Element B. Durch Erosion und anschließende Dilation, was in der Bildverarbeitung als “öffnen” bezeichnet wird, verschwinden Objekte und Teile von Objekten, welche das strukturierende Element nicht vollständig aufnehmen können. Objekte aus dem Bild und die Ausläufer an seinen Rändern, die kleiner als der Ball sind, werden mit Hilfe dieser Operationen entfernt.



[GRAFIK]
Abbildung 3
“Erosion und Dilation”
(Quelle: Springer, 1998)

Der Hintergrund des Bildes besteht bei korrekter Einstellung der Schwellenwerte vollständig aus schwarzen und der Ball aus weißen Pixeln. Die Funktion *findNonZero()* sucht nach den weißen Pixeln im Bild. Von der hierdurch erfassten Pixelgruppe kann der Mittelpunkt als Ballposition verwendet werden. Dadurch ist es eine permanente Ortung des Balls gegeben.

7.1.1.2 Kamera

Die Klasse *Camera* ermöglicht durch die Funktion *getImage()* das Abfragen eines neuen Bildes von der Kamera. Die Funktion *setCameraSettings()* wird in ihrem Konstruktor aufgerufen. Sie holt sich aus der *CameraConfig.txt* Datei die voreingestellten Parameter und schickt diese an die Kamera.

```
width->SetValue(cc.width);
height->SetValue(cc.height);
packetsize->SetValue(cc.packetsize);
exposuretime->SetValue(cc.exposureTime);
offsetx->SetValue(cc.offsetx);
offsety->SetValue(cc.offsety);
```

Die Werte *width* und *height* sind für die Höhe und Breite des Bildes zuständig. Mit ihnen wird das Bild genau an die Größe des Kickertisches angepasst. Mit *offsetx* und *offsety* wird sichergestellt, dass der obere und der linke Rand des Bildes genau auf den Kanten des Kickertisches liegt, da die Positionsberechnung innerhalb des Programms darauf angewiesen ist. Ein weiterer Wert ist *packetsize*, welcher in Bytes angegeben wird. Er ist für die übertragenen Paketgrößen verantwortlich. Er sollte 1500 Bytes groß sein. Bei mehr Bytes gibt es bei der Übertragung ein Problem mit der Netzwerkkarte. Des Weiteren wird der Wert *exposureTime* gesetzt, der für die Belichtungszeit steht, welche in Mikrosekunden angegeben wird. Eine zu niedrig oder zu hoch eingestellte Belichtungszeit bewirkt, dass der Ball an farblichem Kontrast zum Spielfeld verliert, was die Erkennung des Balles erschwert. Die oben angesprochenen Werte aus der *CameraConfig.txt* werden an die *PylonViewer* Api übergeben.

Mit *getCameraSettings()* können die eingestellten Parameter von der *PylonViewer* Api abgefragt und angezeigt werden. Die Funktion *threshold()* erstellt die Fenster für die Einstellung der Schwellwerte zur Erkennung des Balls. Die eingestellten Schwellwerte werden in der *CameraConfig.txt* Datei gespeichert. Beim nächsten Start des Programmes werden die Einstellungen somit übernommen.

7.1.2 Control

Unter *2_Control* werden die Motorkommunikation sowie die Stangen- und Tischsteuerung zusammengefasst. Jede Stange besitzt ihren eigenen *Controller*, welcher mit Hilfe seines *Communicators* den jeweiligen Motor anspricht. Alle Stangen und ihre Motorsteuerung werden im *TableController* zusammengefasst.

7.1.2.1 MotorCommunicator

Für beide Motoren (PS01, RS01) ist je eine Kommunikationseinheit vorhanden. Die wesentlichen Funktionen, die diese unabhängig von Motorart besitzen müssen, werden durch ein Interface vorgegeben. Es herrscht Implementationszwang durch pur virtuelle Methode, welche durch das Gleichsetzen mit Null *deklariert* werden.

```
virtual void linearMovement(int position) = 0;
```

Optionale Funktionen, wie die der Rotation, werden hingegen *definiert*, da nicht jeder Motor technisch dazu in der Lage ist zu rotieren.

```
virtual void rotate(int amount){};
```

Die Konfiguration des Motors wird durch die dafür erstellte *MotorConfig* durchgeführt. Mit *port* wird beschrieben, auf welchem Port der PCAN Adapter beim Betriebssystem bekannt ist.

7.1.2.2 RowControl

Auch der Funktionsumfang der Stangenkontrolle wird durch ein Interface vorgegeben. Hier muss die Funktion *moveTo(float y)* implementiert werden. Funktionen wie *up()*, *down()* und *kick(int strength)* können implementiert werden. Wie bereits erwähnt, hält jeder *RowController* ein *MotorCommunicator* für die Kommunikation mit dem entsprechenden Motor. Die *up()* und *down()* Funktionen bieten die Möglichkeit, die Spielfiguren in eine waagerechte Position zu bringen, um vom Computer geschossene Bälle nicht zu blockieren.
Die Schussfunktion wurde in einen separaten Thread ausgelagert, damit der Verlauf der Anwendung für einen Schuss nicht unterbrochen wird. Mit der *isShooting* Variablen wird eine Überlagerung der Threads verhindert.

7.1.2.3 TableControl

Der *TabelController* hält neben den Konfigurationsdateien und dem *BallStatus* die vier *RowController* der jeweiligen Stangen. Wie der Name implizit verrät, handelt es sich bei diesem Modul um die Tischkontrolle. Diese steht in der Software-Hierarchie am weitesten oben. Sie aktualisiert den *BallStatus* und rechnet die Pixelposition in Millimeter um. Hauptgedanke hinter dieser Einheit ist es, einen zentralen Punkt für die Kontrolle des Tisches zu schaffen.

7.1.3 VirtualKicker

Der *VirtualKicker* wurde mit Hilfe der Grafikbibliothek QT erstellt und bietet eine Alternative zur Darstellung des Tisches. Er wurde in erster Linie erstellt, um für die Entwicklung nicht an den Tisch gebunden zu sein. Es lässt sich ein Kickertisch simulieren, mit welchem andere Ballverfolgungsalgorithmen entwickelt und getestet werden können.

7.1.4 Utilities

Unter *Utilities* werden der *Calculator*, das Konfigurationsmanagement, und die *Modules* Klasse zusammengefasst. Der *Calculator* beinhaltet die Positionsberechnung der Stangen und die der Schussbewegung. Die aktuelle Version bietet mit *calcPositionsSimple()* eine Möglichkeit, die Position der Figuren zu berechnen um den Ball zu halten.

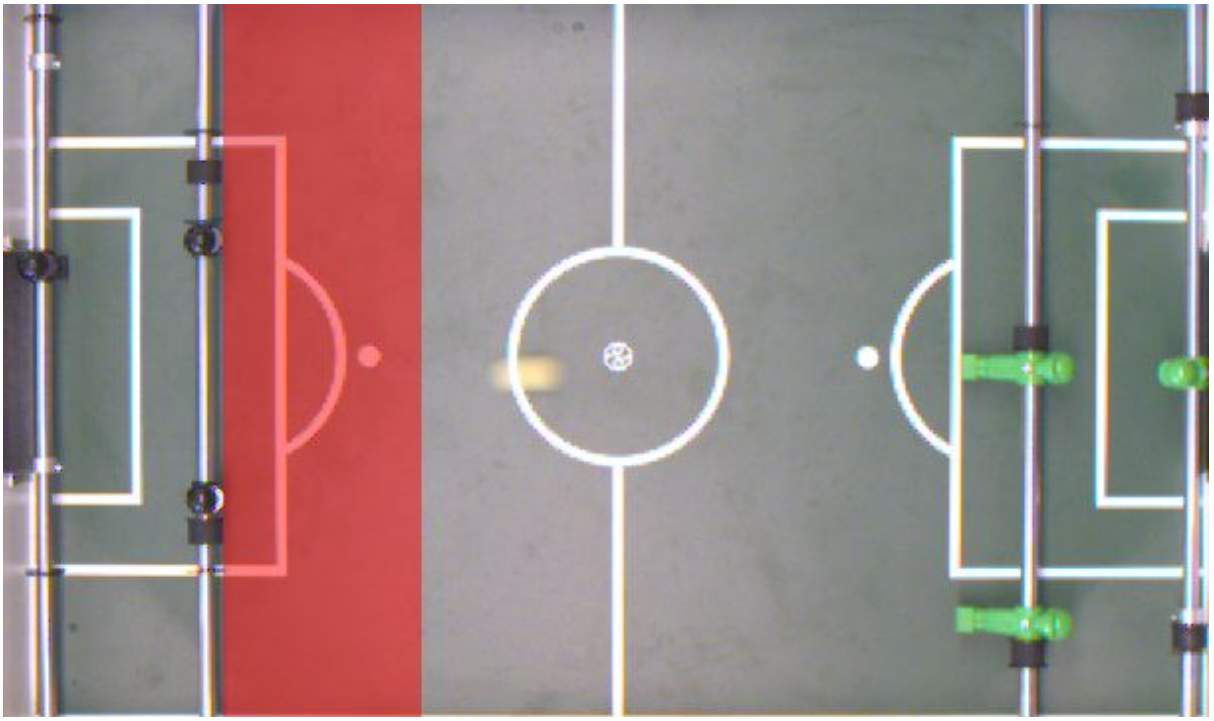
```
static void calcPositionsSimple(std::vector<void*>* params) {
	BallStatus* bs = ((BallStatus*)params->at(0));
	float* positions = ((float*)params->at(1));

	if (tc.isKeeperActive) {
		if ( bs->movement.x < 0 ) {
			float m = bs->movement.y / bs->movement.x;
			m*=(-1);
			float pos = ((bs->position.x - tc.distGoalToKeeper)
* m) + bs->position.y;
			positions[0] = pos;
		}else if ( bs->movement.x > 0) {
			positions[0] = 340;
		}else if (bs->movement.x == 0){
			positions[0] = bs->position.y;
		}
	}
...
```

Diese Funktion arbeitet mit dem aktuellen *BallStatus* und den Positionen der Stangen. Für die Berechnung der Stangenpositionen werden drei Fälle unterschieden. Falls sich der Ball in Richtung des Torwarts bewegt, wird eine Berechnung des Schnittpunktes ausgeführt. Für den Fall, dass der Ball vom Tor des Computers wegrollt, positionieren sich die Figuren mittig. Sollte der Ball sich in x-Richtung gar nicht bewegen, stellen sich die Figuren auf Höhe des Balls. Um die y-Koordinate zu ermitteln, muss zunächst die Steigung *m* des Ballbewegungsvektors berechnet werden. Diese wird dann mit der aktuellen Distanz des Balls zur jeweiligen Spieler-Reihe multipliziert und zum Schluss zur y-Koordinate des Balles addiert.

```
static void calcIfKickSimple(std::vector<void*>* params) {
        BallStatus* bs = ((BallStatus*)params->at(0));
        bool* kick = ((bool*)params->at(3));
        if(bs->position.x < 350 &&
                bs->position.x > 180 && bs->movement.x < 0){
           kick[0] = true;
        } else {
           kick[0] = false;
        }
}
```

Die Berechnung, ob geschossen werden soll oder nicht, erfolgt bisher auf rudimentäre Art und Weise. Die Abwehr führt Schussbewegungen aus, so lange sich der Ball in dem rot markierten Bereich befindet und sich auf die Abwehr zubewegt.



Mit dem *ConfigReader* werden die Konfigurationsdateien ausgelesen und an das Programm weitergeleitet. Das gesamte Framework benötigt eine Vielzahl von Daten. Die Maße des Tisches, die Belichtungszeit der Kamera, die Paketgröße der Bilder und die Beschleunigungswerte der Motoren sind nur einige davon. Alle werden in ihren zugehörigen Konfigurationsdateien gespeichert und zum Programmstart vom *ConfigReader* geladen. Der *ConfigReader* liest bei seiner Instanziierung die an den Konstruktor übergebene Datei vollständig ein, speichert die Daten in einer `std::map<std::string, int>` und bietet anschließend über die Funktionen `std::string getStringValue(const char* name)` und `int getIntValue(const char* name)` die Möglichkeit, diese Werte abzufragen.

Mit *Modules* ist es möglich, vor dem Start des Programms eine Auswahl zu treffen, welche Funktionen verwendet werden sollen. Beispielhaft wurde dieses für den *Calculator* umgesetzt. Die Funktionen des *Calculators* werden zum Programmstart über *Function Pointer* bei der Modules Einheit registriert.

```
string calcPositions calcPositionsSimple
string calcIfKick calcIfKickSimple
```

Innerhalb der *Algorithm.txt* muss festgelegt werden, welche konkreten Algorithmen zum Einsatz kommen sollen. Im Beispiel wird festgelegt, dass zum Zweck der Berechnung der Stangenpositionen die Funktion *calcPositionsSimple()* benutzt wird.

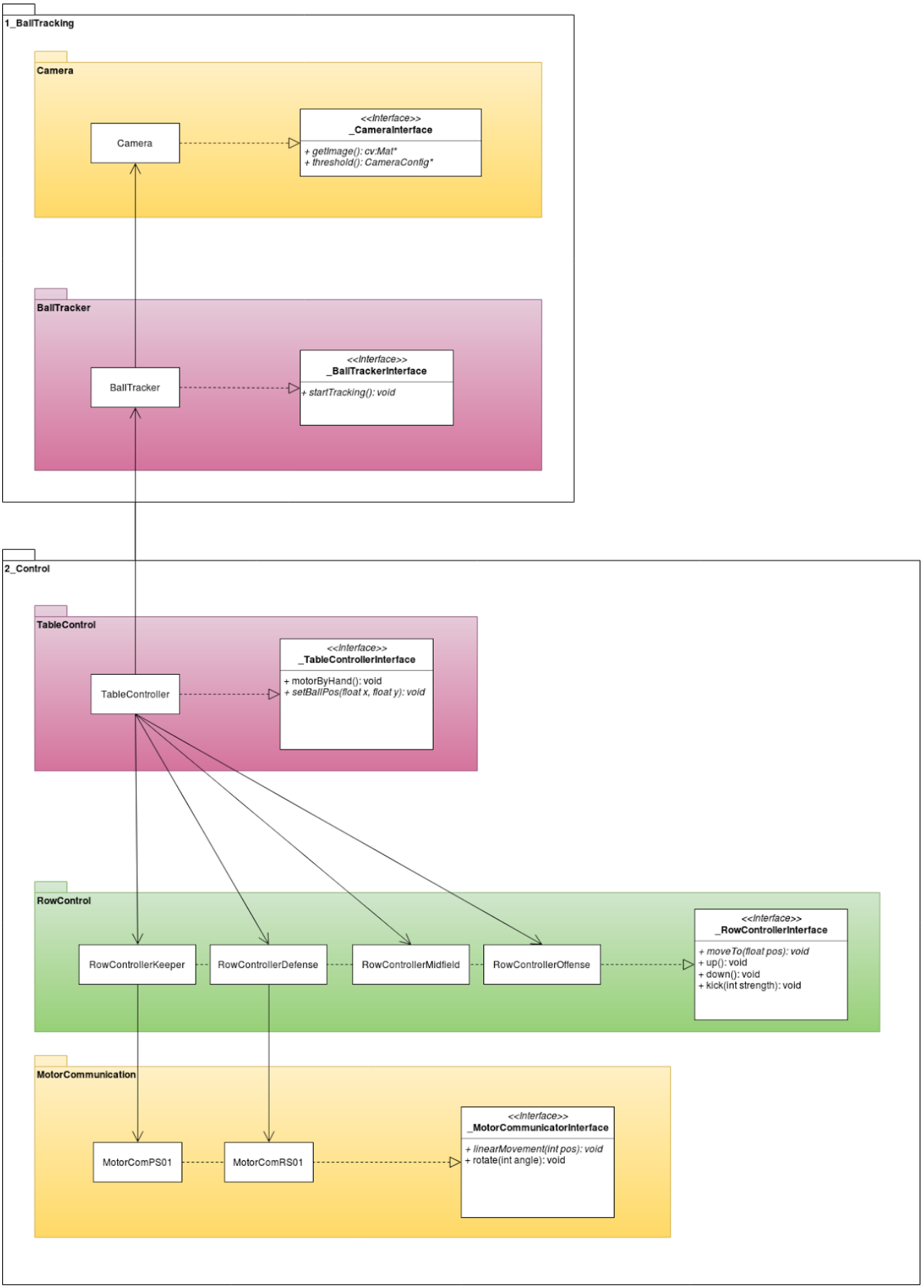
7.1.5 DataType

Es werden einige extra für dieses Projekt erstelle Datentypen genutzt.

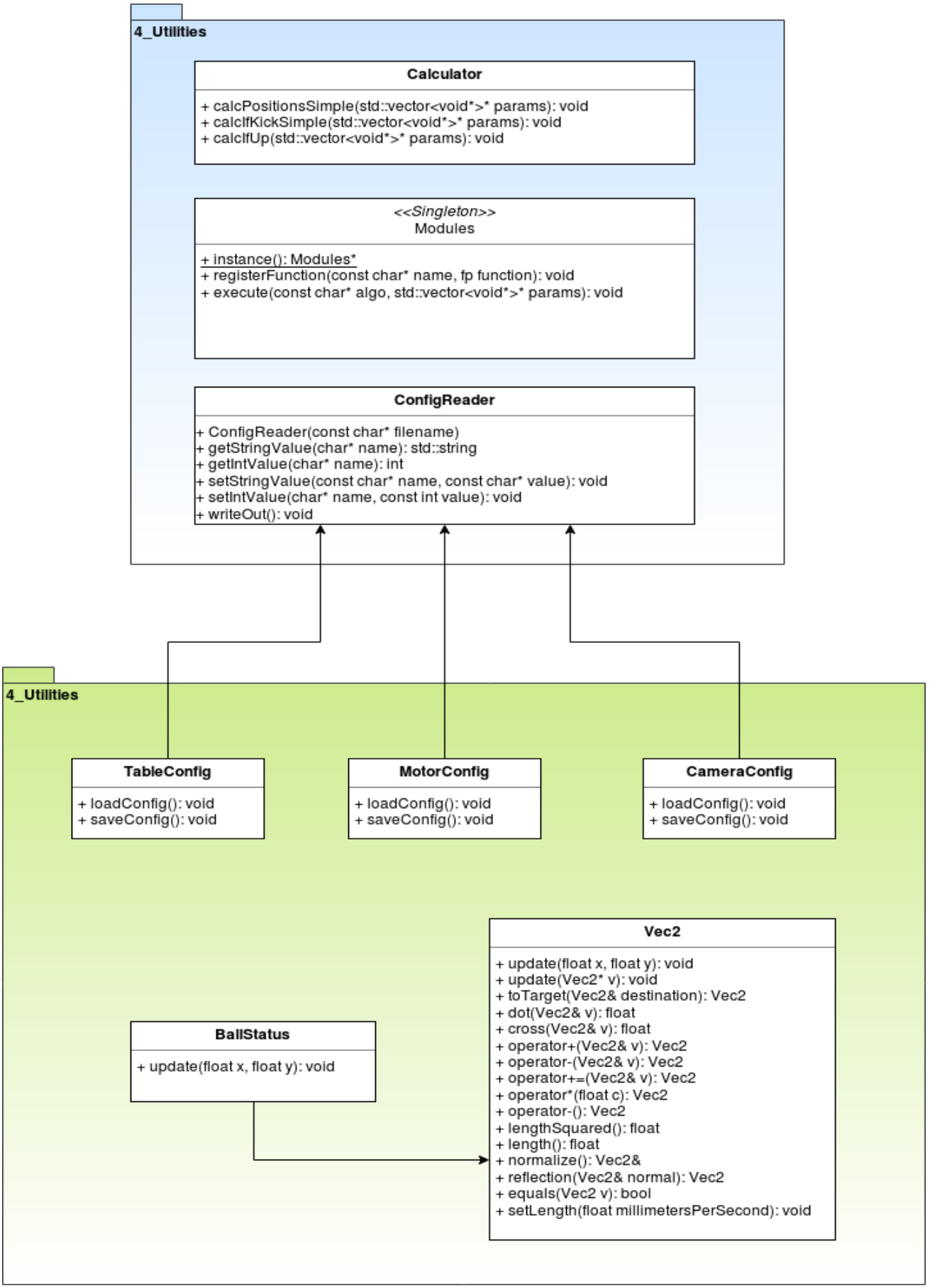
BallStatus.hpp

```
Vec2 movement;
Vec2 position;
```

In ihm wird die aktuelle Ballposition und die Richtung, in die der Ball sich bewegt, festgehalten. Die Länge seines Bewegungsvektors wird in der *update(float x, float y)* Funktion automatisch errechnet und beschreibt seine Geschwindigkeit in Meter pro Sekunde.



[GRAFIK] Abbildung 4: “UML Module”



[GRAFIK] Abbildung 5: “UML Utility”

8 Herausforderungen

8.1 Nibble

Der Nibble dient zur Differenzierung der Signale, die zum Motortreiber geschickt werden. Damit der Motor einen Befehl akzeptiert, muss sich dieser vom vorherigen unterscheiden. Dies wird durch den Nibble sichergestellt. Er ist ein Character Datentyp, welcher initial auf Eins gesetzt ist. Sobald ein Signal an den Motor gesendet wurde, wird von diesem das Komplement gebildet. So hat er abwechselnd den Wert Eins oder Null. Da für den Rotationsmotor zwei Treiber notwendig sind, werden zwei Nibble benötigt. Der nibbleTranslational und der nibbleRotary haben die korrespondierenden Funktionen switchedNibbleT() und switchedNibbleR() für den Wechsel.

Der Nibble spielt bei der Kommunikation mit dem Motor eine wichtige Rolle. Durch ihn wird sichergestellt, dass Befehle deterministisch ausgeführt werden. Wird er nicht beachtet, so kommt es zu einem nicht vorhersehbaren Fehlverhalten des Motors. Es wird an Positionen gefahren, die nicht beabsichtigt sind oder es passiert nichts. Auch die durch das homing erschlossene Grenzen, können durch einen Nibble Fehler überschritten werden, was zu einem Absturz des Motortreibers führt.

8.2 Beschleunigung und Bremsverhalten des Motors

Die Beschleunigung und das Bremsverhalten der Motoren, insbesondere des Motors “Linmot RS01 Hubdreh-Motor”, welcher für die Abwehrstange montiert wurde, sind kraftvoll. Dieser musste im Bezug auf sein Beschleunigungs- und Abbremsverhalten mit einem Bruchteil seiner maximalen Leistungsfähigkeit betrieben werden, da sonst der Aufbau für die Kamera in Schwingungen versetzt wurde. Das Bild verlor dadurch an Schärfe und der Ball wurde unter Umständen an einer Position erfasst, an der er sich tatsächlich nicht befand. Die hierdurch in die Irre geführte Ballerkennung, errechnet eine neue Ballposition fernab der vorherigen und die Motoren müssen eine ruckartige Bewegung ausführen, was den Effekt insgesamt weiter verstärkt. Zum Ende des Projektes wurde die Halterung der Kamera, welche zuvor direkt mit dem Kickertisch verbunden war, entfernt und durch Traverse ersetzt, welche von zwei Stativen gehalten wird. Hierdurch ist er physisch von der Kamera entkoppelt und die Schwingungen wirken sich nicht auf die Bilder der Kamera aus. Der Tisch, auf dem die Motoren befestigt sind, muss von dem Kickertisch entkoppelt werden, um zu vermeiden, dass der Rückstoß von den Motoren nicht an den Kickertisch übertragen wird. Danach kann die Geschwindigkeit des Motors problemlos erhöht werden, ohne dass die ganze Konstruktion anfängt stark zu wackeln.

8.3 Lichtverhältnisse und Belichtungszeit

Die Lichtverhältnisse müssen für den Schwellwertalgorithmus, wie in Kapitel 7.3.1 Kamera beschrieben, konstant sein. Die Probleme mit den Lichtverhältnissen kann man sehr wahrscheinlich mit einer automatischen Anpassung an eine geänderte Lichtsituation lösen.

8.4 frameInit()

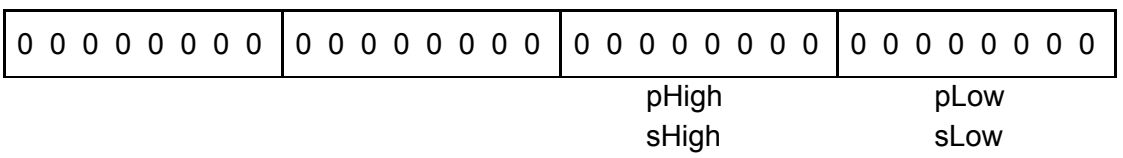
Die *frameInit()* Funktion ist die Kommunikation mit den Motoren bzw. mit den Treibern. Zunächst wird der Port zur Übertragung mittels *openport()* geöffnet. Sobald dies geschehen ist, wird der struct *can_frame* erstellt, der alle zu übertragenen Informationen hält. Dieser wird dann mit *sendPort(&frame)* übermittelt, bevor der Port mit *closePort()* wieder geschlossen wird. Der Frame, der übertragen wird, besteht aus zehn Werten, die eingangs als Paramenter in der *frameInit()* Funktion übergeben werden. Die einzelnen Framedaten und ihre informationen werden nachfolgend anhand eines Beispiels beschrieben. Das Beispiel stammt aus der *linearMovement()* Funktion des *MotorCommunicators* für den PS01.

```
frameInit(0x201, 0x8, 0x3F, 0x00, this->switchedNibbleT(), 0x09, pos1,
pos2, sLow, sHigh);
frameInit(0x301, 0x8, aLow, aHigh, dLow, dHigh, 0x00, 0x00, 0x00, 0x00);
frameInit(0x80, 0x0, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00);
```

Die Informationen werden in zwei Paketen übertragen, jeweils ein *frameInit()* Aufruf. Der dritte dient zur Synchronisation mit dem Motor, die nach jeder vollständigen Datenübertragung aufgerufen werden muss. Aufgrund des simplen Aufbaus der Synchronisation wird auf diese nachfolgend nicht näher eingegangen. Anzumerken ist jedoch, dass die Synchronisation mit 0x80 im ersten Parameter kenntlich gemacht wird. Der restliche Frame wird mit Nullen gefüllt.

Im ersten, sowie im zweiten Teil der Frame Initialisierung, gibt der erste Parameter darüber Auskunft, um welchen Teil der Nachricht es sich handelt, sowie an welchen Treiber gesendet werden soll. Der zweite Parameter gibt an, wie viele Daten übertragen werden. In unserem Fall sind es acht. Nun folgen die acht Framedaten. Beim ersten *frameInit()* Aufruf ist der fünfte Parameter zu beachten. Dieser enthält den Nibble, der hier bereits umgekehrt von der Funktion *switchedNibble()* zurückgegeben wird. Die letzten vier Werte geben jeweils die niedrigsten bzw. zweitniedrigsten acht Bit der Position und der maximalen Geschwindigkeit an. Der erstgenannte Wert beinhaltet immer die niedrigsten acht Bit.

int



Bei dem zweiten *frameInit()* Aufruf verhält es sich ähnlich. Auch hier stehen die ersten beiden Stellen für Frame bzw. Treiber und die Anzahl von Datensätzen. Der Unterschied ist, dass die ersten vier der acht Daten die Beschleunigung und Verzögerung repräsentieren. Auch jeweils wie in dem Beispiel oben, zuerst die niedrigsten acht Bit und darauffolgend die zweitniedrigsten. Damit sind alle Anweisungen an den Motor vorhanden und der restliche Frame wird mit Nullen gefüllt.

Die konstellation der Frame-Initialisierung für die Rotation ist dieselbe. Hier wird im ersten Paket mit den letzten acht Parametern die Rotations Position und Geschwindigkeit angegeben. Im zweiten Paket beinhalten die ersten vier Werte auch wieder Beschleunigung und Verzögerung.

9 Sicherheit

Mit zunehmender Anzahl an Motoren und der hinzugefügten Rotationsfunktion spielt das Thema Sicherheit eine immer wichtigere Rolle. Es lässt sich nicht vermeiden, dass der Ball im Verlauf des Spiels zum Stehen kommt und von Spielfiguren nicht mehr erreicht werden kann. In solch einem Fall tendiert der Spieler gerne dazu einzugreifen und den Ball von Hand wieder ins Rollen zu bringen. Unter Umständen birgt dieses ein Verletzungsrisiko.

Auch das Ende der von den Motoren gesteuerten Stangen stellte zu Beginn eine Gefahr dar. Sie werden auf Beckenhöhe mit hohen Geschwindigkeiten ein- und ausgefahren. Dieses Problem ließ sich jedoch leicht lösen, indem Plexiglasröhren am Ausgang der Stangen montiert wurden. Sie verhindern, dass eine Person in die Laufbahn der Stangen Gerät.

Die Gefahr beim Eingreifen in das Spielfeld konnte jedoch nicht so leicht beseitigt werden. Ein erster Ansatz, dies auf Software Ebene zu lösen, war keine Befehle mehr zum Motor zu senden, falls eine bestimmte Anzahl von erkannten Pixeln bei der Ballverfolgung überschritten wird. Die Kamera erkennt den Ball mittels eines Farbschwellwerts. Wenn, nur der Ball auf dem Spielfeld liegt, wird ein kleiner weißer Punkt erkannt. Wird mit der Hand ins Spielgeschehen eingegriffen, wird die Hand aufgrund der ähnlichen Farbe zum Ball auch erkannt. Die Menge der erkannten Pixel steigt und es werden keine Befehle mehr gesendet. Doch diese Methode ist nicht vollkommen zuverlässig. Ist der Spieler dunkel gekleidet oder hat eine dunklere Hautfarbe, so werden nicht genügend Pixel erkannt.

10 Benutzung

In der main kann zwischen zwei Modi gewählt werden. Der Erste startet nur den virtuellen Kicker. Der Zweite ist der Spielmodus. Wird dieser gestartet, so wird zunächst die Einstellung der Schwellwerte für die Ballerkennung vorgenommen. Es ist darauf zu achten, diese so einzustellen, dass nur noch der Ball erkannt wird. Idealerweise ist der Radius des erkannten Balls möglichst groß, ohne dass noch weitere Bereiche weiß aufflackern. Die Hauptregler zur Einstellung sind *LowB* und *LowG*, ggf. noch *LowR*. Ist das Homing abgeschlossen, kann *Esc* gedrückt und gespielt werden. Liegt der Fokus auf dem QT-Fenster, kann durch Drücken der ‘S’ Taste die Ballgeschwindigkeit angezeigt werden.

11 Setup

Für ein reibungsloses Aufsetzen der Projektumgebung wurde in der readme.txt eine Anleitung erstellt. Internetadressen und Code lassen sich aus ihr passend formatiert entnehmen. Grundvoraussetzung für diese Anleitung ist ein installiertes Betriebssystem, in diesem Fall Ubuntu 16.04.

(1) Begonnen wird mit der Installation der Entwicklungsumgebung Eclipse Neon für C++ Entwickler. Um dies nutzen zu können, benötigt es noch das Java Development Kit (JDK) und die Java Runtime Environment (JRE). Diese lassen sich bequem über die Konsole installieren. Als zentraler Speicherort wurde Github gewählt. Auch hier kann die benötigte Software per Konsole installiert werden, um anschließend das Projekt herunterzuladen.

(2) Um OpenCV für Ballerkennung zu nutzen, benötigt es hier auch einen Download gefolgt von zwei Konsolenbefehlen. Zu beachten ist, dass die Befehle zur Installation aus dem entpackten Verzeichnis ausgeführt werden müssen.

(3) Für die Kamerakalibrierung mit dem Framework muss das Pylon Softwarepaket installiert werden. Die Installationsdateien liegen dem Projekt bei und es müssen auch hier nur ein paar Befehle im Verzeichnis ausgeführt werden.

(4) Nun müssen OpenCV und Pylon noch dem Makefile hinzugefügt werden. Der Code hierzu steht in der Readme Datei. Er muss nach `STRIP = strip` eingefügt werden.

(5) Schließlich müssen nur noch die PCAN Adapter- Einstellungen für die Kommunikation mit den Motoren angepasst werden. Hierzu wird in das `/etc/network` Verzeichnis navigiert. Dort werden mit Hilfe des Editors nano fünf Zeilen hinzugefügt. Nach Speicherung der Änderungen ist das Setup abgeschlossen.

12 Fazit und Ausblick

Bei unserer Projektarbeit kam die Eclipse Version Neon für C++ Entwickler zum Einsatz. Für die meisten Einstellungen sowie Installationen wurde die Konsole genutzt. In der readme.txt wurde eine Anleitung erstellt. Diese dient dem reibungslosen Aufsetzen der Projektumgebung. Für die Versionierung des Frameworks benutzten wir GitHub. Es hat uns ermöglicht, gleichzeitig mit mehreren Personen am Programm zu arbeiten und manchmal auch einige Schritte zurückzugehen, um danach besser und effektiver voranzukommen.

Mit Hilfe der Konfigurationsdateien ist es möglich, alle benötigten Werte des Programms separat abzuspeichern. So können sie bei Programmstart geladen und verwendet werden. Es wurden vier Konfigurationsdateien erstellt: *CameraConfig.txt* für die Einstellungen der Kamera, *MotorConfig.txt* für die Motoren, *Algorithm.txt*. Hier kann festgelegt werden, welche Module verwendet werden sollen und *TableConfig.txt*. Darin stehen alle Informationen über Tischabmessungen. Im Kopf jeder dieser Dateien steht eine kurze Beschreibung und auf was bei der Verwendung zu achten ist.

Zentral über dem Spielfeld wurde eine Kamera montiert, die Bilddaten an den PC weiterleitet. Diese wurde über die PylonViewer-App, welche vorher Installiert wurde, in Betrieb genommen. Einige Einstellungen, die am Anfang des Projekts in der PylonViewer-Software eingestellt werden mussten und für die Kameraeinstellung gebraucht werden, sind in das Framework integriert worden. Dadurch werden sie durch das Framework automatisch eingestellt und verwendet. Der Ball wird dabei mittels der freien Bibliothek erkannt, sodass die dazugehörigen Koordinaten für den weiteren Spielverlauf genutzt werden konnten. Der neu installierte Motor, der sich auf einem Beistelltisch befindet, kann sowohl Drehbewegungen, als auch Linearbewegungen ausführen.

Das in der Anforderungsanalyse beschriebene Ziel der Erweiterbarkeit, Wiederverwendbarkeit und Wartbarkeit ist insofern erreicht, dass sowohl Schnittstellen zur Steuerung der Stangen, als auch zur Echtzeitanalyse der Bilddaten geschaffen. Somit ist die Austauschbarkeit und Erweiterbarkeit sowohl von Algorithmen, als auch von Hardware gegeben.

Mittels der Klasse Modules und der dazugehörigen Konfigurationsdatei *Algorithm.txt* können vorkompilierte Programmabschnitte im Programm ausgetauscht werden, ohne den Quellcode verändern und neu kompilieren zu müssen.

Durch unseren Algorithmus zur Ballerkennung, welcher auf einen ausreichenden Farbkontrast zwischen dem Ball und dem Spielfeld angewiesen ist, mussten wir einige Rückschläge in Kauf nehmen, die durch fluktuierende Lichtverhältnisse hervorgerufen wurden. Wenn der Kickertisch in einem Umfeld mit unbeständigen Lichtverhältnissen betrieben werden soll, müsste das Programm vorher um eine dynamische Anpassung an die Beleuchtung erweitert werden. Bei der weiteren Entwicklung sollte auch auf die GigE-Kabel der Kategorie Cat-6 oder höher umgerüstet werden, da dies von der Firma Basler für die Kamera empfohlen wird.

In den nächsten Projekten, die am automatisierten Kickertisch durchgeführt werden, wird der Kickertisch um weitere vier Stangen erweitert. Dadurch wird eine Ausarbeitung der Spielstrategien umso wichtiger. Zum Beispiel sollen die Spielfiguren beim Blocken des Balls nicht alle hintereinander stehen, sondern etwas versetzt. Dadurch ist eine höhere Chance gegeben, den Schuss zu halten. Beim Angriff sollte der Computer seine Spielreihen hochklappen, um für eigene Schüsse eine freie Schussbahn zu gewährleisten. Es ist auch möglich, die Spielstrategie des Gegners zu erkennen, um diese im weiteren Spielverlauf zu berücksichtigen. Das Framework bietet Schnittstellen zu den genannten Zwecken für schnelle und einfache Einbindung neuer Algorithmen und weiterer Motoren. Auch an der Sicherheit der Spieler sollte noch weiterhin gearbeitet werden. Das Problem der Verletzungsgefahr beim Spielen konnte zum Teil softwaretechnisch behoben werden.

Das endgültige Ziel ist es, einen voll funktionsfähigen autonomen Kicker-Roboter zu entwickeln, der nicht zu schlagen ist! Obwohl das Projekt arbeits- und zeitintensiv war, hat uns die Arbeit im Team gut gefallen und bot sowohl Herausforderungen, als auch Erfolgserlebnisse, die uns bereichert haben. Wir sind mit unserem Arbeitsergebnis zufrieden. Das gesetzte Ziel haben wir erreicht. Dieses Projekt können wir weiterempfehlen und sind uns sicher, dass die nachfolgenden Arbeitsgruppen mit dem bisherigen Ergebnis eine gute Grundlage für die Weiterarbeit am Kickertisch haben.

13 Literaturverzeichnis

Schnattinger, Thomas (2006): “KiRo - Tischfußball gegen den Roboter.” URL: http://www.informatik.uni-ulm.de/ki/Edu/Proseminare/KI/SS06/Ausarbeitungen/08-Schnattinger.pdf (Stand: 04.09.2017).

Seck, Rainer (2011): “Projektbeschreibung Computerkicker (ProCK)” URL: http://kicker.ee.hm.edu/wiki/images/Projektbeschreibung_prock_14102011.pdf (Stand: 04.09.2017).

Hochschule München (2011): Abbildung 1. “Computerkicker (ProCK)”. hm.edu. URL: http://www.ee.hm.edu/forschung/projekte/publikationdetail_196.de.html (Stand: 06.09.2017)

fudder (2006): Abbildung 2. “StarKick”. fudder.de. URL: http://fudder.de/mensch-gegen-maschine-der-freiburger-tischfussball-roboter--118643067.html (Stand: 06.09.2017)

Soille, Pierre (1998): Abbildung 3. “Erosion und Dilation”. Morphologische Bildverarbeitung, Springer.