Hochschule für angewandte Wissenschaften Kempten

**Rocky Hockey IV**

*BILD MIT NEUEM GESTELL*

*Abschlussbericht*

*Entwicklung eines Air Hockey Roboters*

[Fakultät](http://lukas-mertens.de/) Informatik

[Sommersemester](http://lukas-mertens.de/) 2020  
  
  
  
  
07.07.2020

|  |
| --- |
| Zusammenfassung |
| Das Projekt Rocky Hockey ist ein DV-Projekt, das unter der Leitung von Professor Arnulf Deinzer für Studierende im 6. Semester des Studiengangs Informatik angeboten wird. Ziel des Projekts ist es einen Roboter zu Entwickeln, der gegen einen Menschen in einem “Airhockey Match“ antreten kann. |

|  |  |  |
| --- | --- | --- |
| Projekt-Team | | |
| *Name* | *Kontakt* | *Matrikelnummer* |
| *Roman Wecker* | roman.w.wecker@stud.hs-kempten.de | 360659 |
| *Alexander Fichtl* | alexander.m.fichtl@stud.hs-kempten.de | 364931 |
| Thomas Brücklmayr | thomas.bruecklmayr@stud.hs-kempten.de | 369884 |
| Marco Schwärzler | marco.k.schwaerzler@stud.hs-kempten.de | 326906 |
| André Fischer | andre.p.fischer@stud.hs-kempten.de | 366231 |
| Thomas Gantner | thomas.gantner@stud.hs-Kempten.de | 366212 |
| Meiko Mehnert | meiko.mehnert@stud.hs-kempten.de | 360589 |
| Sarah Hörmann | sarah.m.hoermann@stud.hs-kempten.de | 366941 |
| David Lippert | david.c.lippert@stud.hs-kempten.de | 363233 |
| Dominik Veith | dominik.veith@stud.hs-kempten.de | 374134 |
| Projekt-Betreuung | |
| Prof. Dr. Arnulf Deinzer | arnulf.deinzer@hs-kempten.de |

Inhaltsverzeichnis

[Zusammenfassung 1](#_gjdgxs)

[Abbildungsverzeichnis 3](#_30j0zll)

[1. Projekt Ist-Zustand 4](#_1fob9te)

[1.1 IST – Zustand Software 4](#_2et92p0)

[1.2 IST – Zustand Hardware 5](#_tyjcwt)

[2. Codeänderungen 6](#_3dy6vkm)

[2.1 Dokumentation und Softwarearchitektur der JAVA GUI 6](#_1t3h5sf)

[2.2 Neue Implementierungen 6](#_35nkun2)

[2.3 Integration der ArduCam Module 6](#_1ksv4uv)

[3. Hardwareänderungen 8](#_44sinio)

[3.1 Hebefunktion der Tischplatte 8](#_2jxsxqh)

[3.2 Neues Kameragestell 8](#_z337ya)

[3.3 ArduCam Module 8](#_3j2qqm3)

[3.4 Elektrik und Verkabelung 8](#_1y810tw)

[3.5 Aufbau der Zwischenplatte 8](#_4i7ojhp)

[4. Probleme 9](#_2xcytpi)

[5. Ausblick Team V 10](#_1ci93xb)

[6. Anleitungen und Tipps 11](#_3whwml4)

[Literaturverzeichnis 12](#_2bn6wsx)

# Abbildungsverzeichnis

Abbildung 1: ABBILDUNGSBEISPIEL: Natürliche Nervenzelle (Quelle: Lukas Mertens, 2018) [4](#_3znysh7)

Abbildung 2: Display mit Java GUI (Quelle: Eigene Abbildung) [7](#_2s8eyo1)

Abbildung 3: Sensor der Lichtschranke (Quelle: Eigene Abbildung) [7](#_17dp8vu)

Abbildung 4: Raspberry Pi 3 Pins (Quelle: raspberrypi.org/documentation/usage/gpio, 2020) [7](#_3rdcrjn)

Abbildung 5: Schaltplan der verwendeten Module (Quelle: Eigene Abbildung) [8](#_26in1rg)

Abbildung 6: MVC Pattern (Quelle: de.wikipedia.org/wiki/Model\_View\_Controller, 2010) [8](#_lnxbz9)

# 1. Projekt Ist-Zustand

1.1 Einleitung

Es ist ein Nachbau eines Air-Hockey Tisches, wie man ihn aus Spielhallen kennt, gegeben. Dieser Nachbau besteht aus einem Gehäuse, in der sich die Elektronik des Tischs versteckt, eine Spielplatte zum Herausheben und einem Gestell für eine Festmontierte Kamera sowie einem Anzeigebildschirm für den Punktestand.

Eine Seite des Tisches kann von einem menschlichem Spieler bespielt werden, wohingegen die andere Seite durch einen festmontierten Roboter gesteuert wird. Der Roboter verwendet die über dem Tisch montierte Kamera. Die Kamera überblickt die Spielhälfte des Roboters und erkennt den Puck auf der Spielplatte. Die Kamera gibt das Bild weiter an die Software die aus dem gewonnen Bild neben der Flugbahn auch die Geschwindigkeit des Pucks sowie mögliche Verteidigungspositionen berechnet.



*Abbildung 1: ABBILDUNGSBEISPIEL: Natürliche Nervenzelle (Quelle: Lukas Mertens, 2018)*

## IST – Zustand Software

Die Software des Projekts umfasst C# Module sowie eine JAVA – GUI

Unser Repository:

<https://github.com/R-Studios/Rocky-Hockey-IV>

🡪 Sollten glaube dem Prof auch einen Zugang geben damit das würde die Übergabe zum nächsten Projektteam deutlich beschleunigen und wir hätten nicht so viel Stress im nächsten Jahr

1.1.1 C# - Abschnitte

GoalDetectionFramework:   
Erfasst die Tore, sobald die Pucks die Lichtschranken überschritten haben.

MotionCaptureFramework übermittelt einen Stream der Kamera. Aus den gewonnen Bilder wird über das MoveCalculationFramework die Flugbahn sowie die Geschwindigkeit des Pucks berechnet und gleichzeitig eine Verteidigungsstrategie berechnet. Im MovementFramework werden die Motoren angesteuert die jeweiligen Befehle für die Bewegungen stammen hierbei aus dem MoveCalculationFramework.

1.1.2 JAVA – GUI

Die verwendete GUI, die über den Monitor am Gestell verwendet wird0

1.1.3 Repository

Der Projektcode wurde uns in mehreren Branches über eine Dropbox übergeben. Ein bestehendes Repository war nicht vorhanden

## IST – Zustand Hardware

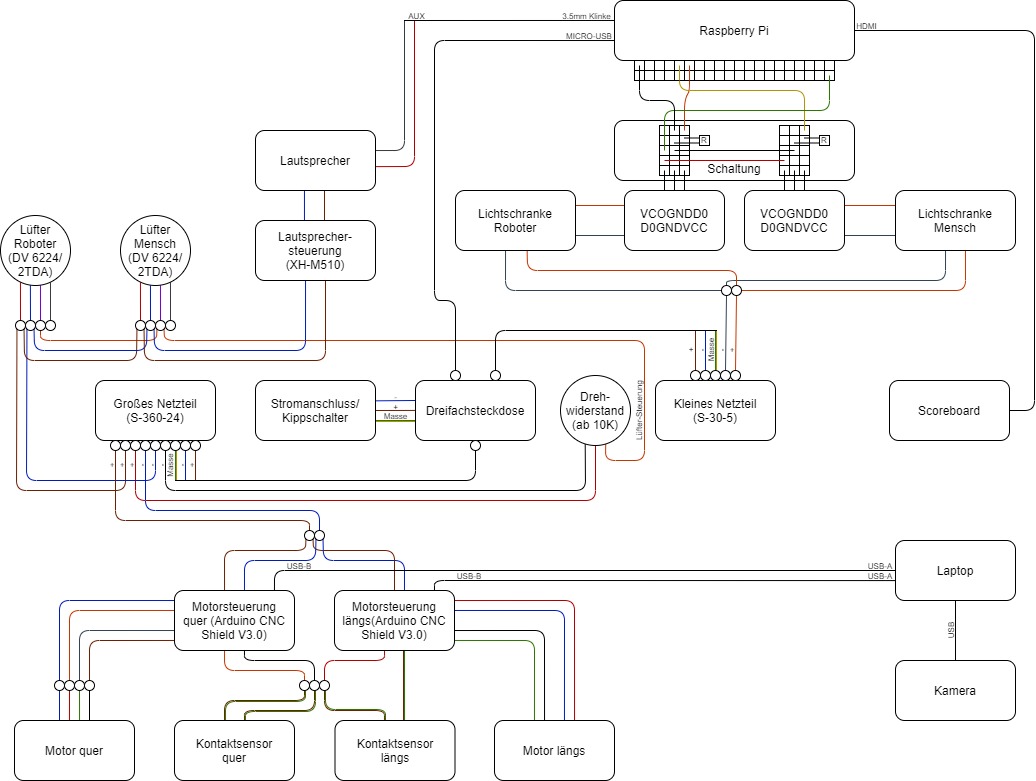
**Hardwarebeschreibung:**

Über einen Kippschalter, der sich an der Unterseite des Tisches befindet, kann die Stromzufuhr gesteuert werden. Der Anschluss erfolgt über ein gewöhnliches Kaltgerätekabel, welches in einen weiblichen Kaltgeräteanschluss gesteckt wird. Am anderen Ende dieses Anschlusses befindet sich eine Dreifachsteckdose, an der zwei Netzteile und die Stromversorgung des Raspberry Pi angeschlossen sind. Das größere Netzteil versorgt die beiden Lüfter, die Motorsteuerungen, die Lautsprechersteuerung und einen Drehwiderstand, über den die Geschwindigkeit der Lüfter reguliert werden kann. Die Lichtschranken beziehen den Strom vom kleineren Netzteil. Das Raspberry Pi ist mit einem USB Netzteil über ein Micro-USB Kabel angeschlossen. Das Scoreboard muss extern mit einem Micro-USB Anschluss über eine Powerbank oder ein weiteres USB-Netzteil versorgt werden.

Das Raspberry Pi ist mit dem Lautsprecher über einen Chinch zu 3,5mm Klinke Kabel verbunden. Außerdem ist das Scoreboard über ein HDMI Kabel angeschlossen. Über die Pins des Raspberry Pi´s werden die beiden Lichtschranken angesteuert. Die genaue Belegung und weitere Details folgen in einem späteren Abschnitt der Dokumentation.

Die Motorsteuerungen sind jeweils mit einem USB-A zu USB-B Kabel an den Laptop angeschlossen, auf dem die benötigten Frameworks laufen. Des Weiteren sind die Motoren über 4 Pole und die Kontaktsensoren des Roboters über 2 Pole mit dem jeweiligen Board verbunden.

Um eine bessere Übersicht zu erhalten, wurde folgender Schaltplan erstellt:



*Abbildung 2: Schaltplan (Quelle: Eigene Abbildung)*

**Netzteile:**

Für die Stromversorgung der Komponenten werden diverse Netzteile verwendet. Das größere Netzteil ist vom Typ S-360-24 (AC Input: 110/240V +-15%; DC Output: +24V 15A; Power: 360W) und das kleinere ist vom Typ S-30-5 (AC Input: 110/240V +-15%; DC Output: 5V 6A). Das Raspberry Pi und das Scoreboard sind über 2 USB-Netzteile angeschlossen.

**Lüfter:**

Für das typische Gleiten beim Air-Hockey sorgen zwei 90W ebm-papst Industrie Lüfter (DV 6224/2TDA). Zu finden sind diese auf der unteren Seite des Tisches.

**Drehwiderstand:**

Über ein Draht-Potentiometer (AB Elektronik 10k) wird die Geschwindigkeit der Lüfter gesteuert.

**Raspberry Pi:**

Der Raspberry Pi übernimmt die Darstellung der GUI und ermittelt die Tore über die Laserschranken.

**Monitor:**

Über den Monitor wird eine GUI abgerufen über die man ein Spiel starten kann. Während eines laufenden Spiels zeigt der Monitor außerdem den Punktestand an.

**Laserschranken:**

Laserschranken in beiden Toren zeichnen auf, ob ein Tor geschossen worden ist. Sie sind über zwei Vibrationsmodule (D0GNDVCC/VCOGNDD0) an das Raspberry Pi angebunden.

**Motorsteuerungen:**

Zwei Arduino CNC Shield V3.0 sind für die Steuerung der Motoren und der Kontaktsensoren zuständig.

**Motoren:**

Die Bewegungen des Roboters werden über zwei Nema Stepper Motoren ermöglicht.

**Kontaktsensoren:**

Jeder Achse des Roboters besitzt einen Kontaktsensor, über den die Motorsteuerung erkennen kann, wann das Ende der Fahrbahn erreicht ist.

**Lautsprecher:**

Über ein Soundboard werden Spielsounds (z.B. bei einem Tor) abgespielt.

**Lautsprechersteuerung:**

Das Verstärker-Board XH-M510 bereitet den Sound vom Raspberry Pi für den Lautsprecher auf.

**Kamera:**

Bei der Kamera handelt es sich um eine Playstation 3 Eye Kamera, die via Motion-Tracking das Spiel aufzeichnet. Die Kamera befindet sich an einem Gerüst auf der Spielseite des Roboters.

# 2. Codeänderungen

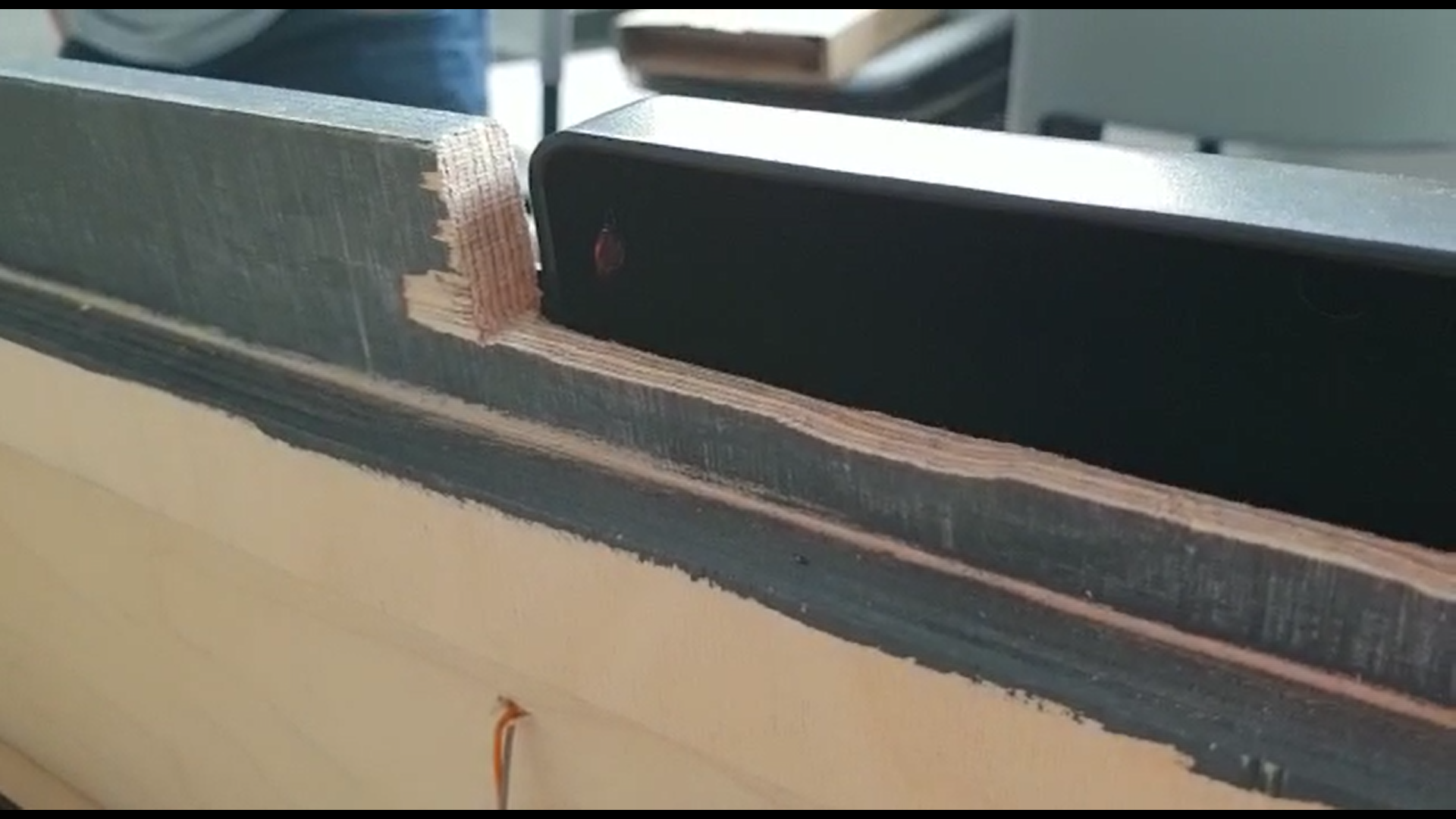
## 2.1 Dokumentation und Softwarearchitektur der JAVA GUI

Die grafische Benutzeroberfläche des Air Hockey Tischs läuft auf einem Raspberry Pi 3 Model B+ und wird über HDMI auf einem Display, welches am Tisch montiert ist für den Spieler dargestellt.

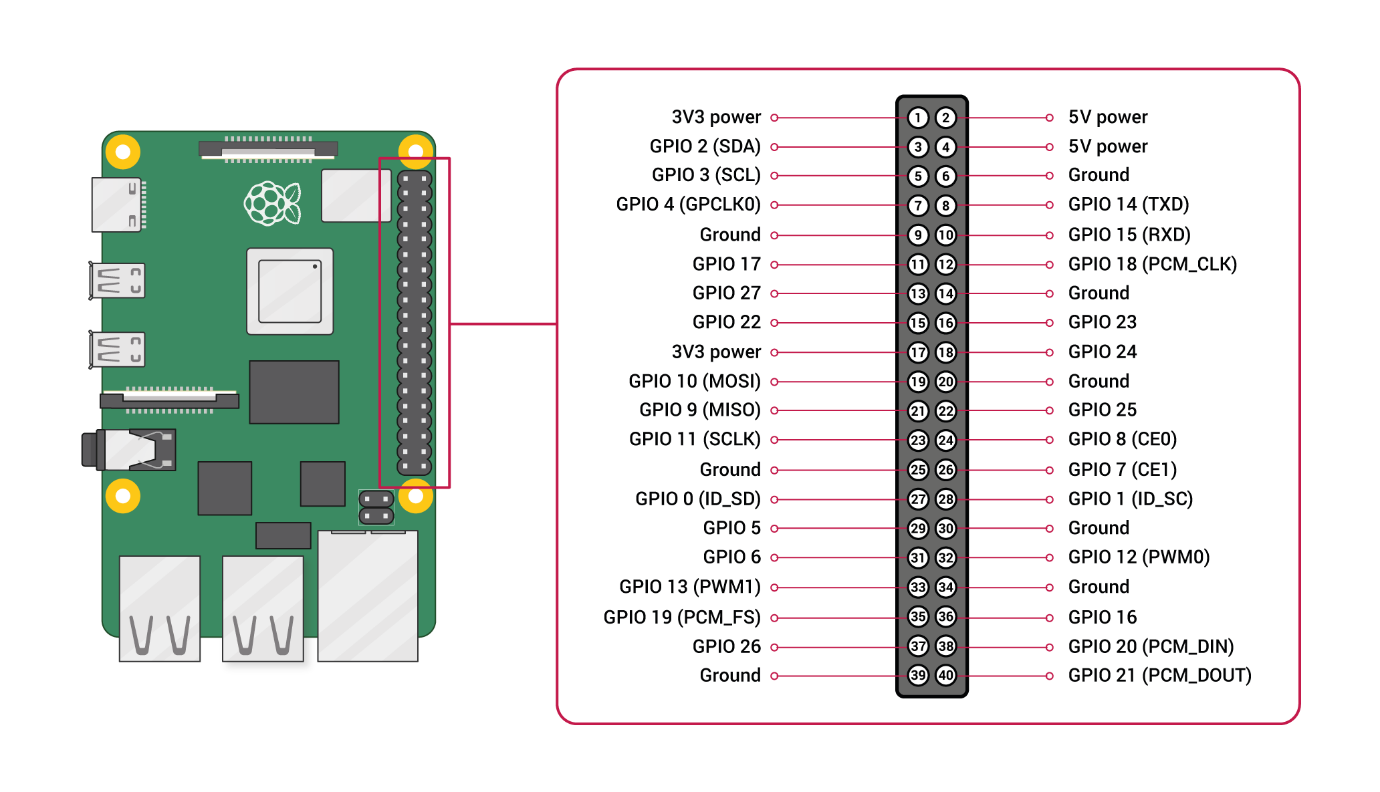


*Abbildung 3: Display mit Java GUI (Quelle: Eigene Abbildung)*

Die Oberfläche zeigt die Scores des Spielers und des Roboters, sowie die verbleibende Spielzeit am oberen Bildrand an. Es gibt Optionen zum Starten, Neustarten, Stummschalten und Beenden des Spiels. Die Tore werden über zwei Lichtschranken gezählt, welche im Tor verbaut sind.



*Abbildung 4: Sensor der Lichtschranke (Quelle: Eigene Abbildung)*

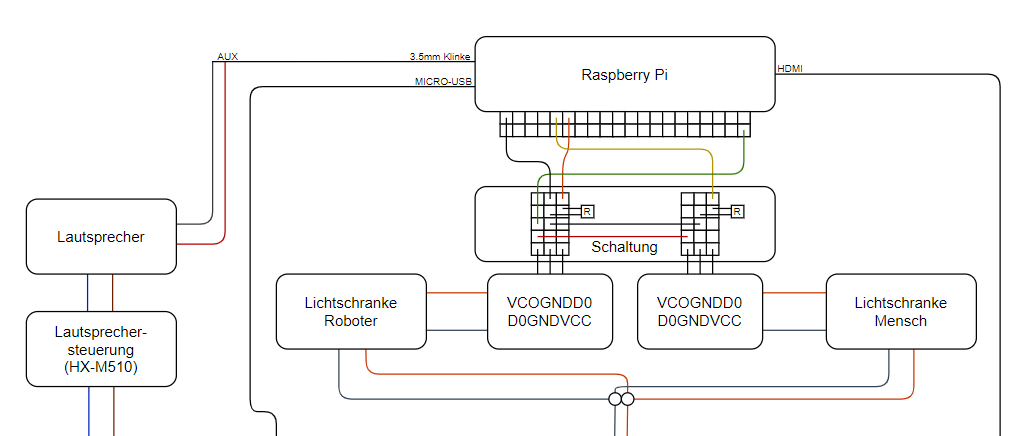


*Abbildung 5: Raspberry Pi 3 Pins (Quelle: raspberrypi.org/documentation/usage/gpio, 2020)*

Verwendete Pins:

Pin 2 (5V power), Pin 29 (GPIO 5), Pin 31 (GPIO 6), Pin 39 (Ground)

Über die GPIO Pins 5 und 6 sind die beiden Lichtschranken verbunden.



*Abbildung 6: Schaltplan der Anschlüsse des Raspberry Pi´s (Quelle: Eigene Abbildung)*

Die Controller der Lichtschranken sind über ein Breadboard mit dem Raspberry Pi verbunden.

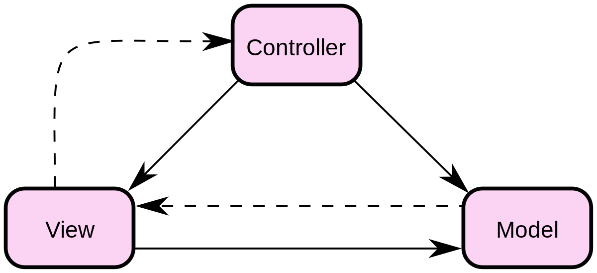
Hardware Troubleshooting:

* kein Sound: Lautsprecher muss über AUX Klinkenstecker verbunden sein und Audio muss auf “analog” in Raspbian umgestellt werden
* kein Bild: Raspberry Pi muss über HDMI mit dem Display verbunden sein und der Raspberry Pi und das Display müssen mit Strom versorgt werden (empfohlen 5V / 2,5 A)

Raspberry muss nach Start kurz grün aufleuchten, sonst ist die Spannung zu gering

* keine Interaktion möglich: Maus und Tastatur über USB Ports verbinden (kein Touchscreen)

Die Benutzeroberfläche wurde in Java Swing entwickelt und nach dem Model-View-Controller Software-Pattern umgesetzt. Die Hauptklasse “Main” erzeugt ein “Controller” Objekt, welches Singleton-Objekte für die Klassen “Gui”, “Audio” und “HardwareIO” erzeugt.



*Abbildung 7: MVC Pattern (Quelle: de.wikipedia.org/wiki/Model\_View\_Controller, 2010)*

Die Klasse “Audio” erzeugt mehrere “AudioThreads” damit Sounds gleichzeitig abgespielt werden können (zum Beispiel Hintergrundmusik und Scoresounds). Die Klasse “ResourceLoader” ist eine Hilfsklasse die Bilder und Audioquellen als InputStream lädt. Diese wird dafür benötigt um Ressourcen in der exportierten .jar Datei zu laden.

Die Java GUI wird als .jar Datei exportiert und in das Verzeichnis /home/pi abgelegt. Es wurde ein Autostart Script für die JavaGUI.jar erstellt. Solange der Name der Datei gleich bleibt, funktioniert das Autostart Script weiterhin, wenn die GUI überarbeitet wurde. Nach einem Neustart öffnet sich die GUI nach einem Delay von 10 Sekunden automatisch.  
Manuell kann sie mit dem Befehl java -jar JavaGUI.jar über das Terminal ausgeführt werden.

Die Java GUI wurde vollständig mit Hilfe von JavaDoc dokumentiert:  
<https://github.com/R-Studios/Rocky-Hockey-IV/tree/master/JavaGUI/doc>

## 2.2 Neue Implementierungen

**Ein Vorwort zum C#-Projekt**

Die Gesamtaufgabe "Airhockey-Roboter" ist in mehrere Unterbereiche (Verantwortungen) auf verschiedene Komponenten aufgeteilt. Diese Komponenten können zur Erfüllung ihrer Aufgaben auf den Ergebnissen anderer Komponenten aufbauen. Zum Verständnis dieses Kapitels werden hier einige Komponenten kurz erklärt:

ImageProvider: Stellt Bilder einer Kamera bereit.

PositionCollector: Ermittelt Positionen des Pucks inklusive Zeitstempel der Abfrage. Die Standardimplementierung verwendet dazu Bildmaterial eines ImageProviders.

MovementController: Steuert Motoren über serielle Ports an.

PathPrediction: Berechnet den Bewegungsweg des Pucks anhand der Koordinaten die von einem PositionCollector zurückgegeben werden.

StrategyManager: Wählt eine passende Strategie aus. Die Entscheidung basiert auf der PathPrediction.

TrajectoryCalculationFramework: Verwaltet MovementController, PathPrediction und StrategyManager und stellt Verbindungen zwischen den Komponenten her.

Zu Beginn des Projektes wurden einige Zeilen Code überarbeitet, um die Performance des MoveCalculationFramework zu verbessern, außerdem musste an vielen Stellen die Dokumenation des Codes verbessert werden, da hier nur ungenügende Kommentare vorhanden waren und Methoden und Vorgehensweisen nicht deutlich genug gekennzeichnet worden sind. Zudem wurden einige Dateien und Methoden die nicht dienlich waren aus dem Projekt entfernt.

**Hinzufügen einer Kamera-Vorschau**

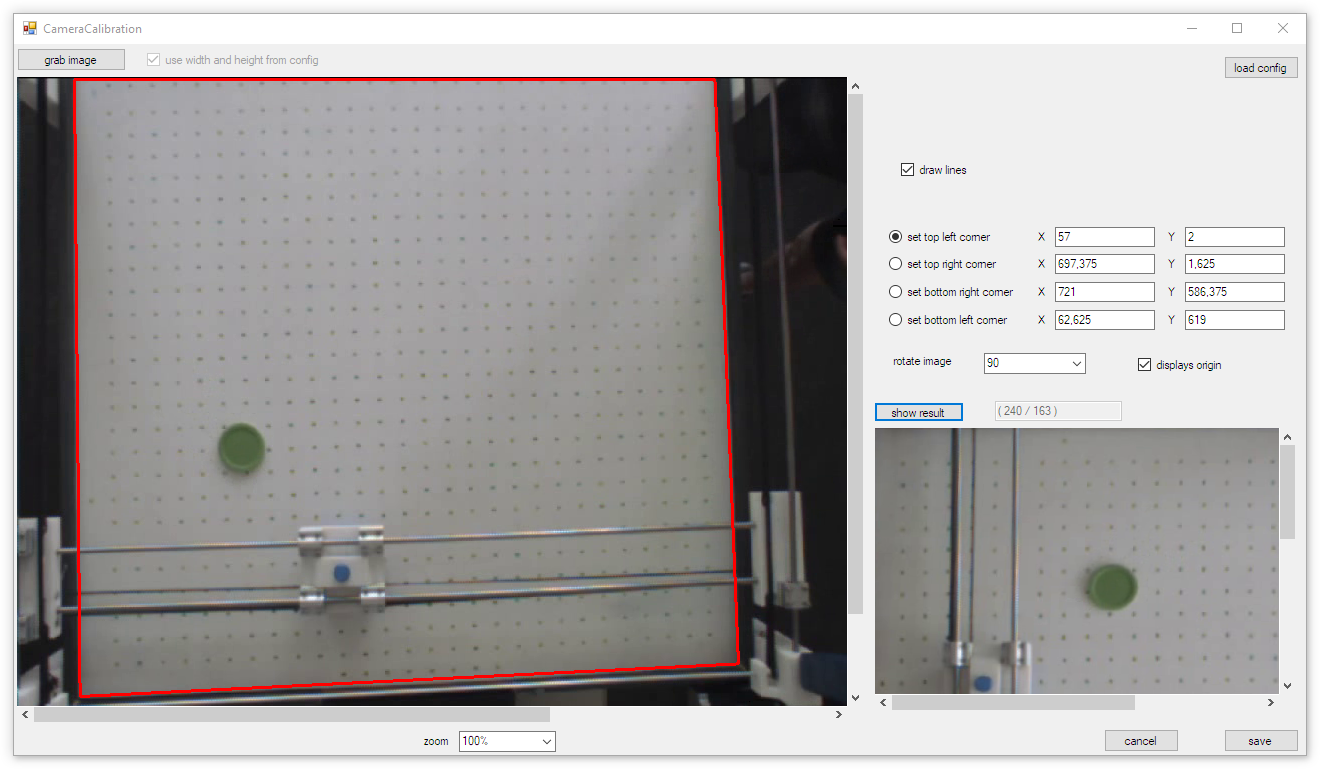
Unser Team hatte Schwierigkeiten, den Tisch überhaupt zum Laufen zu bekommen. Oft war dabei nicht klar, an welcher Stelle der Aufbau überhaupt fehlschlägt. Damit man den Status des Bots besser nachvollziehen kann, wurde in das Hauptfenster der RockyHockeyGUI ein Kamera-Panel eingefügt. Dieses zeigt nachdem der Bot gestartet wird ein Live-Bild der verwendeten Kamera an. Auf dieselbe Kamera kann aber nicht mehrmals zugegriffen werden. Deshalb wurde bei der Umstrukturierung des Codes der Kamera-ImageProvider, welcher die Bilder für die Puckpositionserkennung bereitstellt, erweitert. Er speichert sich dazu das zuletzt erfasste Bild zwischen und stellt dieses dann für das Hauptfenster zur Verfügung.

***< evtl. Bild vom Hauptprogramm mit Bildmaterial der EyeToy Kamera einfügen? >***

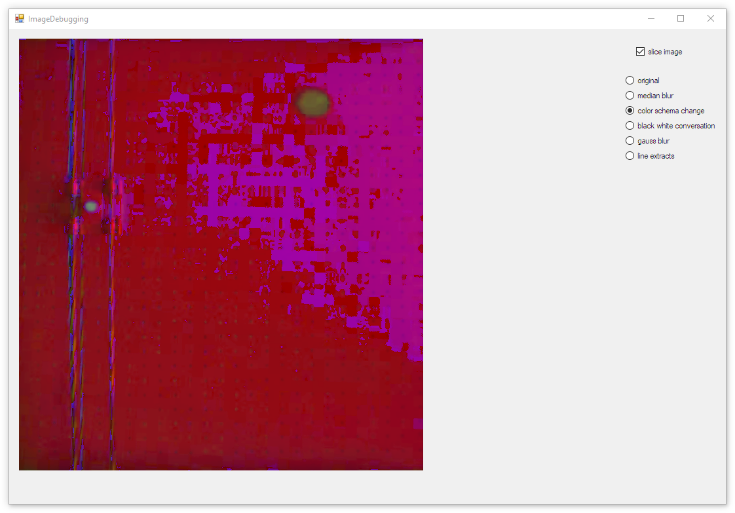
**Positionserkennung**

Um die Positionserkennung zu verbessern und zu kalibrieren wurden drei Fenster erstellt.

**CameraCalibration-Fenster**

Da auf dem Bild der Kamera unter Umständen viel Fehlinformationen sind, zum Beispiel wenn noch Bereiche außerhalb des Spielfeldes aufgenommen werden, oder das Bild möglicherweise perspektivisch verzerrt ist, wurde dieses Fenster erstellt

In dieses Fenster kann man ein aktuelles Bild der Kamera laden. In diesem Bild wird danach markiert, wo auf dem Bild das Spielfeld zu sehen ist. Diese Markierung wird mittels vier Punkten vorgenommen, die in das Bild eingezeichnet werden. Um diesen Vorgang zu erleichtern kann nötigenfalls in das Bild vergrößert werden. Hat man die vier Punkte festgelegt, kann man eine Vorschau angezeigt lassen. Bei bedarf kann man das Ergebnis des Entzerrens / des Zuschneidens auch noch um jeweils 90° drehen.

**ImageDebugging-Fenster**

Dieses Fenster ermöglicht es, die unterschiedlichen Verarbeitungsstadien des Bildes anzuzeigen. Auch kann man das Bild vor und nach der Entzerrung / dem Zuschneiden anzeigen lassen. Dies ermöglicht es, eventuelle Fehler in de Bilderkennung, zum Beispiel mit der Kalibrierung der Kamera, ausfindig zu machen

**CircleDetectionCalibration-Fenster**

Da am Anfang keine oder falsche Positionen für den Puck erkannt wurden, hat man dieses Fenster erstellt.

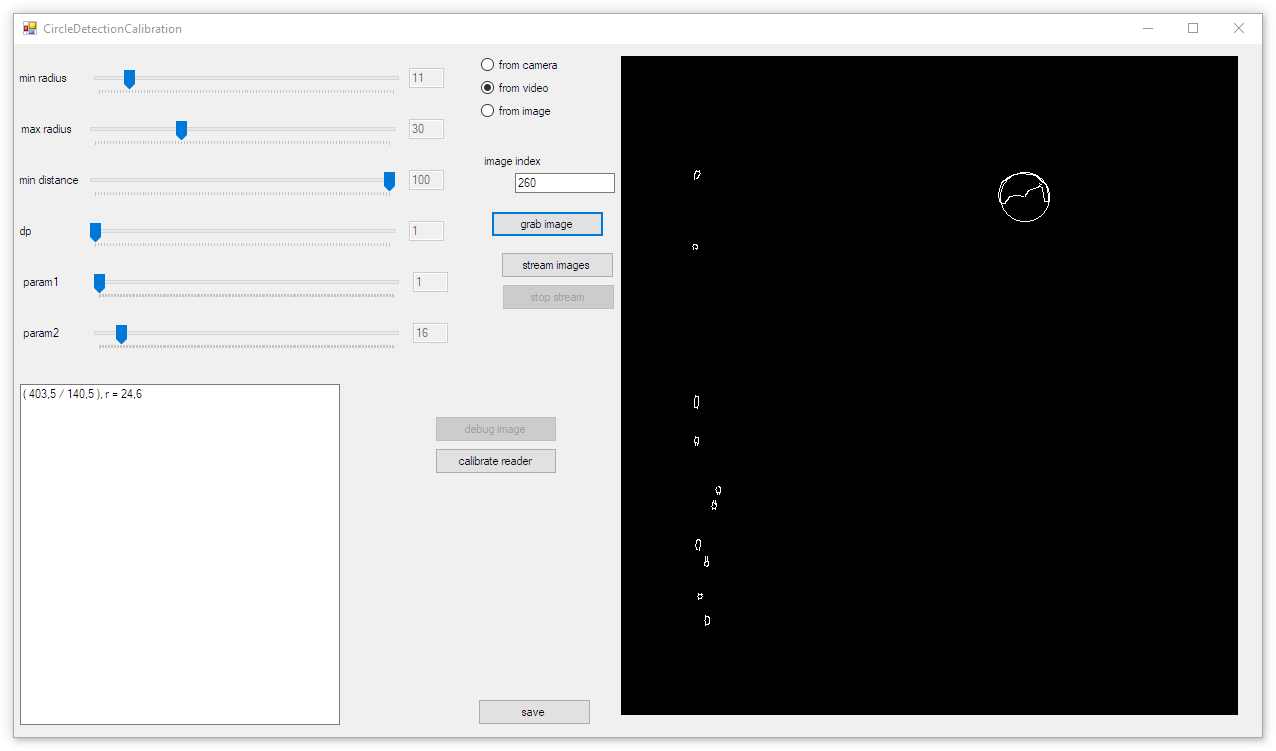
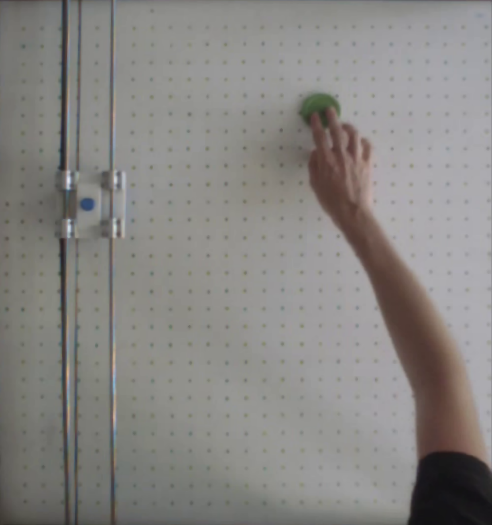


Abbildung : Original des Bildes vor der Verarbeitung

In diesem Fenster kann man die Parameter für die Kreiserkennung auf dem einstellen. Min Radius ist der minimale Radius, den Kreise haben müssen, um erkannt zu werden. Max Radius ist der maximale Radius, den Kreise haben dürfen, damit sie erkannt werden. Min Distance ist der minimale Abstand zwischen den Mittelpunkten zweier erkannter Kreise. Param1 und Param2 sind so benannt, da die Parameter, die der Verarbeitungsmethode übergeben werden, ebenfalls so heißen. Der Erkennungsmethode wird auch ein Algorithmus, besser gesagt ein Identifier für einen Algorithmus übergeben. Param1 und Param2 sind algorithmusspezifische Parameter und somit ist keine allgemein gültige Benennung möglich, da die Funktion der Parameter in unterschiedlichen Algorithmen unterschiedlich sein kann. Im verwendeten Algorithmus stellt Param2 die minimale Größe ein, die ein Objekt auf einem Bild haben muss, damit kreise um das Objekt herum erkannt werden können.

**Pfadvorhersagungs-Komponente**

Die Klasse PathPrediction ermöglicht es, den Zeitpunkt zu ermitteln, wann der Puck einen bestimmten Punkt erreicht. Um das zu bestimmen, wird der Klasse bei der Instanziierung zunächst ein Objekt übergeben, das den PositionCollector implementiert. Beim Aufruf der Methode init() der PathPrediction Instanz, wird von diesem PositionCollector eine Liste mit Positionen inklusive Zeitstempel abgefragt. Aus diesen Positionen wird die Flugbahn und Geschwindigkeit des Pucks errechnet. Um die Daten aktuell zu halten wird anschließend ein Aktualisierungs-Thread gestartet. Dieser holt sich immer die aktuelle Position des PositionCollecors und gleicht diese mit der berechneten Flugbahn ab und aktualisiert die Puckgeschwindigkeit. Sollte sich die aktuelle Postion trotz Toleranz nicht auf der berechneten Flugbahn befinden, wird die Aktualisierung gestoppt und mittels init() die Vorhersage reinitialisiert.

**Implementierung einer Strategienkomponente**

Da der bisherige Code nur darauf aus war die Flugbahn des Pucks zu berechnen und auf die berechnete Flugbahn lediglich mit einem Block zu antworten haben wir uns entschlossen, dass MoveCalculationFramework dahingehend zu verändern, dass der Roboter auch aggressivere Manöver ausführen kann. Die Strategie Komponente besteht aktuell aus vier Teilen. Eine StrategyManager Klasse, einem StrategyCommunicationInterface Interface, einer Implementierung dieses Interfaces, die Klasse SimpleStrategie und der Klasse DirectionForStrategy. Der Ablauf der Strategieberechnung ist folgender:

Zuerst wird eine Position bestimmt, an der der Puck geblockt werden soll, falls keine berechnete Strategie eingesetzt werden kann. Dort wird der Schläger anschließend hin bewegt. Danach wird eine Anzahl Instanzen von Strategieklassen erstellt, momentan vier. Diesen Instanzen werden über Methoden des Interfaces vom StrategyManager gewählte x-Koordinaten übergeben und die Berechnung der Strategien angestoßen. Die jeweiligen Berechnungen der Strategien verlaufen parallel zu einander. Die übergebenen x-Koordinaten geben an, in welchem Abstand zur Seite des Roboters der Puck gespielt werden soll. Für die Berechnung der Strategie wird durch die PathPrediction Instanz die x-Koordinate in eine Position mit Zeitstempel umgewandelt. Dabei prüft die PathPrediction lediglich, wann laut Vorhersage die x-Koordinate erreicht wird und berechnet Zeitpunkt und zugehörige y-Koordinate Nach dieser Berechnung wird ein Vektor zurück gegeben, der die Richtung des Pucks zu Zeitpunkt des Schlages angibt und dessen Endpunkt die berechnete Position für gegebene die x-Koordinate ist. Nachdem die Strategieklasse diesen Vektor erhalten hat, wird die Richtung bestimmt, die der Puck nach dem Schlag haben soll. Dies erfolgt mittels der statischen Klasse DirectionForStrategy. Der Methode getPuckDirection dieser Klasse werden vier Parameter übergeben. Die Startposition des Pucks, die Endposition des Pucks, wie oft der Puck über Bande gespielt werden soll und auf welcher Längsseite der Puck zuerst abprallen soll. Aus diesen Daten wird die Richtung berechnet, die der Puck nach dem Schlag benötigt, damit die vorgesehene Flugbahn ausgeführt wird. Diese vier Parameter werden von der SimpleStrategie Klasse zufällig gewählt. Aus der Richtung des Pucks vor dem Schlage und der Richtung des Pucks nach dem Schlag, wird die Bewegung berechnet, die der Schläger braucht, um den Puck in die neue Richtung auszulenken. Mittels dieser Richtung werden zwei Schlägerpositionen errechnet, eine Position vor dem Schlag und die Position nach dem Schlag. Daraufhin wird noch berechnet, wie lange das ausführen dieser Strategie brauchen würde und die Berechnung ist abgeschlossen. Nachdem alle Strategieberechnungen abgeschlossen sind, wählt der StrategyManager eine davon aus. Dies erfolgt dadurch, dass geprüft wird, ob noch genug zeit vorhanden ist eine der berechneten Strategien auszuführen. Ist dies der Fall wird die entsprechende Strategie benachrichtigt und diese gibt die berechneten Schlägerpositionen an den MovementControler weiter. Wenn keine zeit übrig war, ein der Strategien auszuführen, so wird ein Flag gesetzt. Sobald sich die Pfadvorhersage aktualisiert, weil der Puck am Schläger des Roboters oder am Rand abgeprallt ist und das Flag gesetzt ist, werden nochmals Strategien erstellt, um den Puck zurück zu schlagen, die wiederum auf Ausführbarkeit getestet werden. Unabhängig vom Ausgang dieses Tests ist der Berechnungszyklus danach abgeschlossen.

**Bau eines virtuellen Tischs**

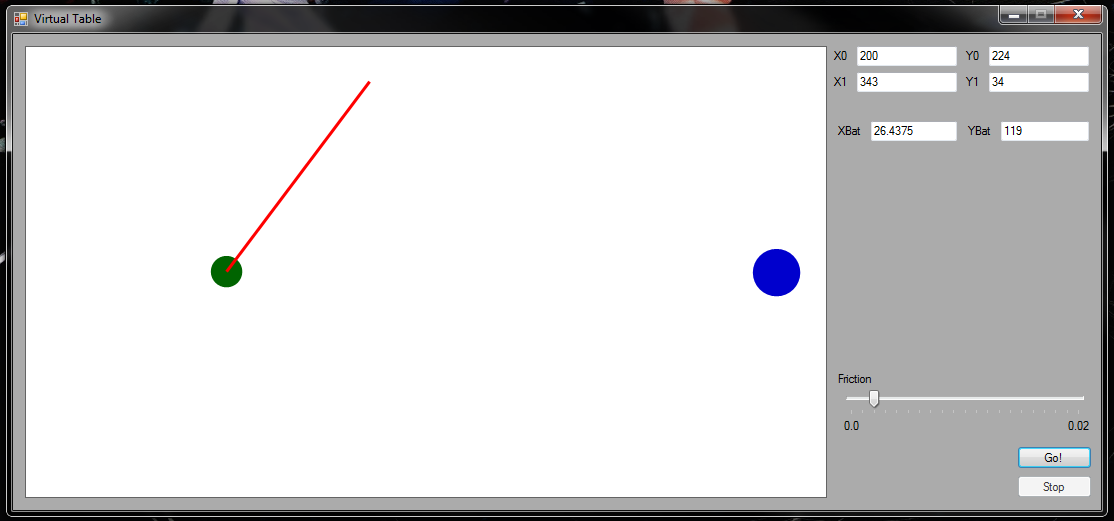
Die Anpassungen an den Komponenten für Bilderfassung, Positionserkennung, Pfadberechnung und Bewegungsstrategien wurden hauptsächlich mit Unterstützung durch Unittests gemacht. In einer Besprechung ist dann aber die Idee aufgekommen, einen virtuellen Tisch zu erstellen um die Pfadberechnung und Bewegungsstrategien interaktiv testen zu können, auch ohne Zugang zum realen Tisch.

Der virtuelle Tisch ist von der Größe dem physischen Tisch nachmodelliert. Puck und Schläger sind ebenfalls maßstabsgetreu abgebildet. Die Oberfläche kann direkt aus dem Hauptfenster der RockyHockeyGUI geöffnet werden. Sie öffnet sich in einem eigenen Fenster und meldet sich als alternativer PositionCollector. Wird nun der Bot gestartet, verwendet das TrajectoryCalculationFramework den virtuellen Tisch statt der Standardimplementierung für die Positionsermittlung. Während der Puck still liegt, kann er in der Oberfläche per Klick verschoben werden. Zieht man eine Linie, wird diese als Bewegungsvektor gemerkt. Wenn auf den Start-Button geklickt wird, beginnt die Simulation und der Puck gleitet über das Spielfeld. Mit dem Stop-Button wird der Puck sofort gestoppt und kann danach wieder verschoben werden.

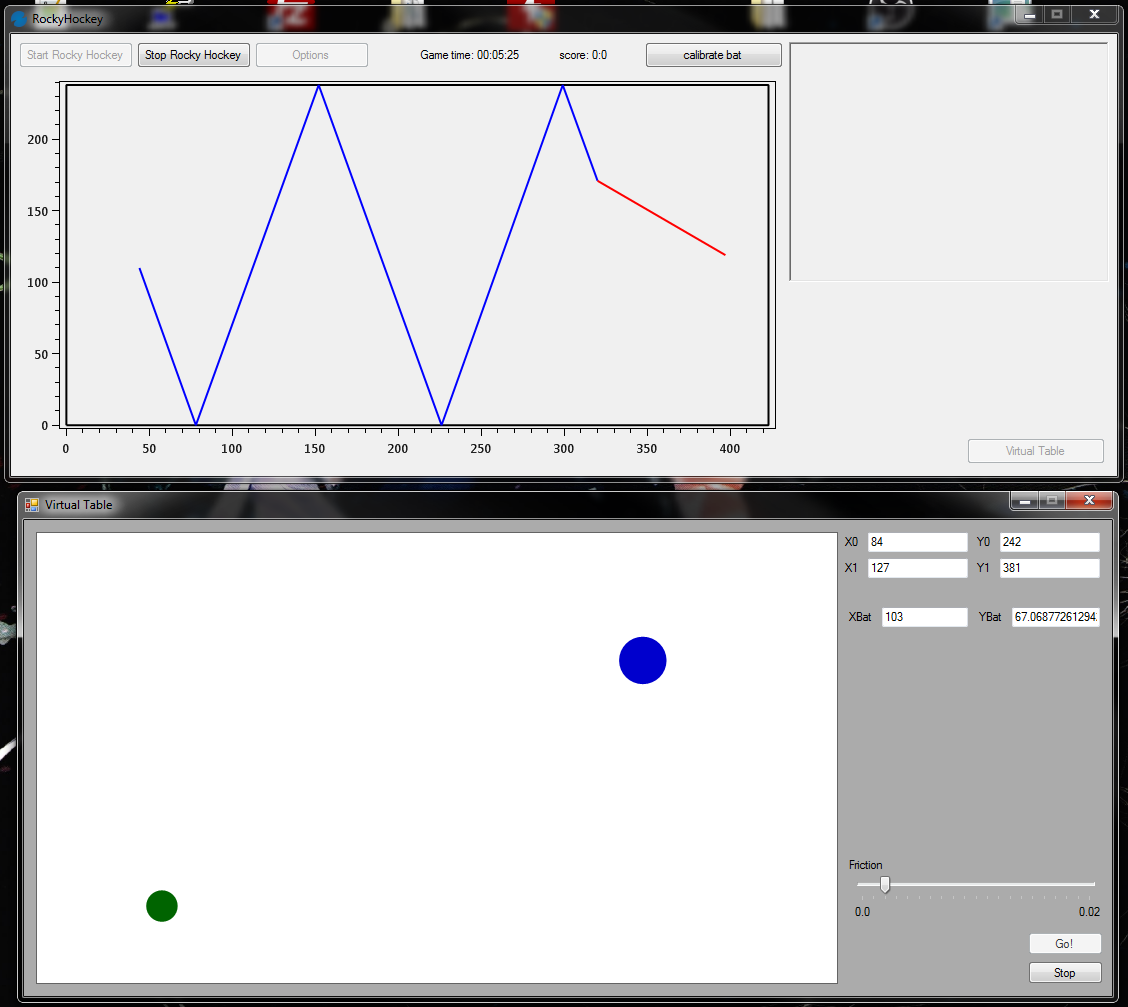
Entscheidet sich die Strategie, den Schläger zu bewegen, so werden die Bewegungsdaten an den MovementController weitergegeben. An dieser Stelle im MovementController wurde ein Event eingefügt, sodass der virtuelle Tisch die Bewegungsziele abfangen und selbst darstellen kann.

Der Tisch simuliert Reibung und Abprallen des Pucks an den Banden. Diese Mechaniken sind nur Annäherungen an die realen physikalischen Prozesse, die allerdings für unsere Zwecke ausreichend sind.

Dadurch, dass mit Windows Forms nur 30 FPS erreicht werden können, findet die Simulation des Tischs in einem eigenen Thread statt. Da mehrere Threads eventuell gleichzeitig auf den Zustand des Tischs zugreifen möchten - UI-Thread des Fensters, MovementController, viele PathPrediction-Tasks - war ein Synchronisationsmechanismus nötig. Der Zustand des virtuellen Tischs ist nach außen nicht sichtbar. Allerdings kann eine lesende oder schreibende Funktion an einen Zugriffspunkt übergeben werden, welcher dann atomaren Zugriff auf den Tischzustand garantiert.



*Setzen des Geschwindigkeitsvektors*



*Puck in Bewegung; Pfadvorhersage und Schlägerbewegung*

Repository:

Als Repository haben wir in GitHub eine Projektgruppe erstellt für zehn Personen. Dazu wurde das alte Projekt auf einem Branch festgehalten während wir abseits vom Masterbranch auf refactored C# die Änderungen am alten Projekt bearbeiteten. Das Finale Projekt wurde dann über C#-Alpha-Version weiterentwickelt ehe es zum Schluss auf den Master gepusht wurde.

## 2.3 Integration der ArduCam Module

\*\*\* Alexander Fichtl \*\*\*

# 3. Hardwareänderungen

## 3.1 Hebefunktion der Tischplatte

\*\*\* Andre Fischer – Hebefunktion der Tischplatte \*\*\*

## 3.2 Neues Kameragestell

**Vorher:**

Das alte Kameragestell war relativ instabil und für die unten dafür vorgesehenen Halterungen zu klein, wodurch es in alle Richtungen gewackelt hat. Aus diesen Gründen haben wir uns dazu entschieden eine neue zu bauen.

*Abbildung 10: Kameragestell vorher (Quelle: Eigene Abbildung)*

**Nachher:**

Die neue Kamerahaltung passt gut in die unten vorgesehenen Halterungen, da 4cm x 6 cm Latten dafür verwendet wurden. Die mittleren Latten führten nicht zu einer besseren Stabilität, weshalb sie weggelassen wurden. Zur Verbesserung der Stabilität wurden stattdessen parallel zur Länge des Tisches Latten angebracht.

*Abbildung 11: Kameragestell nacher vorne (Quelle: Eigene Abbildung)*

*Abbildung 12: Kameragestell nacher seitlich (Quelle: Eigene Abbildung)*

Die Kamera wird an einer Latte festgeschraubt und damit längs über dem Tisch ausgerichtet um das beste Bild zu bekommen.

*Abbildung 13: Kamerabefestigung (Quelle: Eigene Abbildung)*

## 3.3 ArduCam Module

Seit diesem Semester steht außerdem eine Arducam CMOS AR0135 zur Verfügung, die die Playstatio3 Eye Kamera ersetzen soll.

\*\*\* Alexander Fichtl \*\*\*

## 3.4 Elektrik und Verkabelung

|  |  |
| --- | --- |
|  | **Vorher:**  Die Verkabelung des Tisches war sehr provisorisch gestaltet. Kabel wurden nicht wirklich gebündelt und wurden teilweise nur lose verlegt, was zur Folge hatte, dass sich die Nachverfolgung zur Erkennung der Funktionalität sehr schwierig gestaltet hat. Außerdem waren einige Kabelhalterungen gebrochen. Zur Isolierung einiger Kabel wurde Kreppband verwendet, was in dieser Form keine dauerhafte Lösung darstellt. Eine Dokumentation der Kabelfunktionen und der Anschlüsse war bis dahin noch gar nicht vorhanden. Auch die verwendeten Komponenten waren nicht dokumentiert. |
| *Abbildung 13: Tischverkabelung zuvor (Quelle: Eigene Abbildung)* | |
| **Nachher:**  Zunächst wurde der Schaltplan erstellt, der bereits im Ist-Zusatand der Handware zu sehen ist. Damit werden zukünftige Arbeiten an der Verkabelung des Tisches erleichtert. Anschließend wurden die Komponenten in einem übersichtlichen Layout angebracht. Danach wurden alle Kabel unter der Berücksichtigung ihrer Funktion gebündelt. (z.B. Strom für die Motorsteuerungen oder Datenkabel der Laserschranken) Nach der Bündelung wurden alle Kabel mit passenden Kabelführungen sauber verlegt. Dies sorgt für eine bessere Übersicht der Verkabelung. Dabei wurden auch die Aufsitzpunkte der Zwischenplatte berücksichtigt. Abschließend wurden alle offenen Kontaktstellen mit Isolierband verdeckt, um Kurzschlüsse zu vermeiden. |  |
|  | *Abbildung 14: Tischverkabelung danach (Quelle: Eigene Abbildung)* |

## 3.5 Aufbau der Zwischenplatte

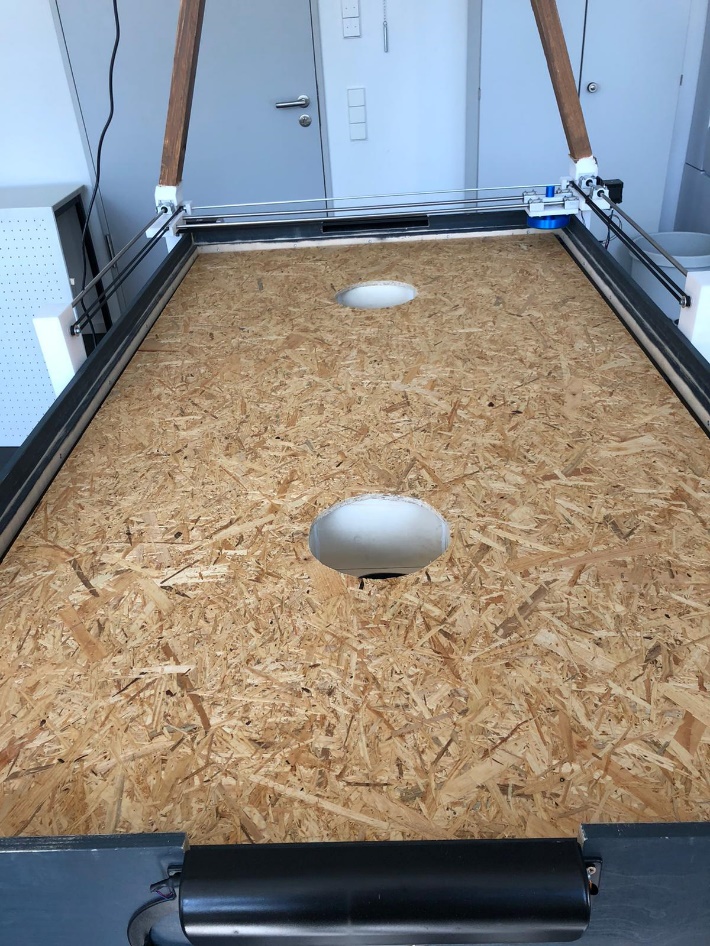
Bei den ersten Tests des Tisches fiel auf, das die Luft durch den Tisch sich nicht gleichmäßig verteilt, was zu einem ungleichmäßigem Gleiten des Pucks führt. Die Vermutung war, dass es daran liegt, dass der Abstand zwischen Lüfter und der Tischplatte zu groß war und dadurch die Luft die Platte nicht ausreichendversorgt. Als erstes wurde überlegt, die Lüfter die am Boden befestigt sind höher zu befestigen. Am ende entschied man sich jedoch eine weitere Zwischenplatte einzubauen.   
Diese besteht aus zwei Löchern an der Position der Lüfter, die über Zylinder mit der unteren Platte verbunden sind. Sie dienen gleichzeitig auch als Standfüße. Dadurch wird die Luft nach oben geleitet und erst direkt unter der oberen Tischplatte verteilt. Dies soll ermöglichen, dass mit geringerer Lüfterleistung und geringerer Lautstärke das gleiche Ergebnis erzielt werden kann, als ohne Zwischenplatte. Aufgrund der Hitze, die beim betreiben des Tisches entsteht wurde an den Seiten Platz gelassen, damit die Luft hier besser zirkulieren kann.

Abbildung 15: Zwischenplatte am Tisch angebracht (Quelle: Eigene Abbildung)

Ein Bild, das Screenshot, Uhr enthält.

Automatisch generierte Beschreibung

Abbildung 13: Maße/Aufgbau der Zwischenplatte (Quelle: Eigene Abbildung)

4. Probleme

\*\*\* ALLE \*\*\*

**Beispielabschnitt**

**Vorheriges Team**

Durch Herrn Prof. Dr. Deinzer war es möglich direkt zu Beginn des Semesters mit dem vorherigen Team in Verbindung zu treten. Jedoch war dabei der Austausch innerhalb der Teams sehr mühselig, da vor allem im vorhergehenden Team gerade einmal zwei ehemalige Studenten dem jetzigen Team zugearbeitet haben und Fragen beantworten konnten. Wir wurden teilweise mit offenen Fragen zurückgelassen und der Info wir sollen uns offene Fragen aus dem Abschlussbericht ausarbeiten. Was jedoch aufgrund der ungenügenden Dokumentation kein Spaß war. Ein Großteil der Anfangsphase war deswegen damit beansprucht, dass vor allem das Programmierteam sich in den Code einlesen musste und Zusammenhänge und Rückschlüsse selbst erarbeiten musste. Erschwerend dazu kam die unzureichende Dokumentation innerhalb des Codes hinzu.

**Virtueller Tisch**

Windows Forms und das darunterliegende GDI sind nicht gut geeignet für flüssige Bewegungen und Animationen. Die Simulationslogik des Tischs läuft jetzt auf einem eigenen Thread und erlaubt es dem Fenster auch mit wenigen FPS zu laufen ohne die Genauigkeit der Positionsermittlung zu beeinträchtigen.

Es sind viele verschiedene Größen und Größenfaktoren überall verteilt definiert und es ist nicht klar was welche Auswirkungen hat. Das macht sich unter anderem bei den neu implementierten Strategien bemerkbar. Die Daten, die an die Motorsteuerung geschickt werden, müssen von Spielfeldposition in mm umgerechnet werden. Der virtuelle Tisch, der dann diese umgerechneten Daten abfängt, muss sie auf seine eigene Größe zurückskalieren. Geht in diesem Ablauf irgendwo eine Umrechnung schief kommen am Ende unbrauchbare Werte heraus.

5. Ausblick Team V

**Raspberry Pi**- Das vorhandene Netzteil für den Raspberry Pi besitzt nicht ausreichend Spannung. Dadurch startet der Raspberry Pi nicht immer, besonders wenn Peripherie angeschlossen ist.

**Display**- Für das Display wird ein neues Netzteil benötigt, vor Ort ist keines vorhanden. Bisher wurde das Display über eine externe Powerbank betrieben.  
- Das Display könnte durch ein Touch-Display ersetzt werden, damit man die GUI ohne externe Peripherie verwenden kann.

**C#-Projekt**

- In der gesamten Projektmappe könnten Positionen und Größen vereinheitlicht werden.

- Der virtuelle Tisch könnte um realistische Schlägerbewegung und Abprallen des Pucks vom Schläger erweitert werden.

**Elektrik und Verkabelung**

- Die verwendeten Kabelfarben sind im Laufe der Zeit sehr uneinheitlich geworden. Hier könnten die Kabel für gleiche Funktionen durch einheitliche Kabel ersetzt werden. (z.B. Rot für Plus-Leitung, usw.)

- Einige Kabel sind durch Zwischenstecker und Isolierband miteinander verbunden. Diese Kabel können entwerder verlötet werden oder durch durchgehende Kabel ersetzt werden.

\*\*\* ALLE \*\*\*

**Beispielabschnitt**

Beispieltext

Hier sollte ausgeschrieben werden, welche Ziele wir verfolgt haben aber nicht beenden konnten. Bzw. welche Ansätze vom nächsten Team verfolgt werden sollten.

Reihenfolge möglichst nach Überschriften in Abschnitt 3 und 4.

6. Anleitungen und Tipps

\*\*\* ALLE \*\*\*

**Beispielabschnitt**

Beispieltext

V

Vollständige Angabe zur Ausarbeitung der ArduCam

Reihenfolge möglichst nach Überschriften in Abschnitt 3 und 4.

**Ausarbeitungen**  
Dokumenationen, Wochenberichte, die Abschlussberichte der Vorgängergruppen, das alte C# - Projekt, die JAVA GUI findet man alles in unserem Repository u.a. auch in den verschiedenen Branches. Zum Ende des Projekts sollte auch Herrn Deinzer ein Zugang dazu ermöglicht werden, damit eine Übergabe deutlich einfacher und schneller abgewickelt werden kann.

**Elektrotechniker**

Veränderungen an der Elektronik benötigen gewisse Erfahrungen. Sollte diese in eurem Team nicht vorhanden versucht doch über den Professor jemanden aus der Fakultät Elektrotechnik für euch zu gewinnen.

# Literaturverzeichnis

**Zitierweise**

Alle Zitate und Belege wurden nach dem APA-Zitierstandard verfasst. Wenn sich längere Abschnitte auf eine Quelle beziehen, wird immer nur einmal im Abschnitt auf die Quelle verwiesen.

Agarap, A. F. M. (2019, 7. Februar). *Deep Learning using Rectified Linear Units (ReLU)*. University of Cornwell, [arXiv:1803.08375](https://arxiv.org/abs/1803.08375) [cs.NE]. Abgerufen 2. Juni, 2019, aus der arXiv Datenbank.