Hochschule für angewandte Wissenschaften Kempten

Rocky Hockey IV

Abschlussbericht

Entwicklung eines Air Hockey Roboters

Fakultät Informatik
Sommersemester 2020

15.07.2020

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 Air Hockey 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

Inhalt

Abbildungsverzeichnis

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 Tisches versteckt, einer Spielplatte zum Herausheben und einem Gestell für eine festmontierte Kamera sowie einem Display für den Punktestand.

Eine Seite des Tisches kann von einem menschlichen Spieler bespielt werden, wohingegen die andere Seite durch einen 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.

1.2 Ist – Zustand Software

Die Software des Projekts umfasst C# Module sowie eine Benutzeroberfläche in Java.

Für das Projekt haben wir ein eigenes GitHub Repository erstellt. Das Projekt unserer Vorgängergruppe befindet sich auf dem Branch „old“. Auf den „refactored“ Branches wurden einzelne Bestandteile des C# Projekts zeitgleich überarbeitet. Das finale C# Projekt befindet sich auf dem Branch „refactored-C#-virtual-table“. Die Java GUI befindet sich auf dem „Master“ Branch. Der Branch „wiki-resources“ enthält sämtliche Dokumentationen des Projektes.

Das Repository:

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

Die nächste Gruppe kann sich das GitHub Projekt forken, um auf unserem Stand aufzubauen.

1.2.1 C# - Abschnitte

GoalDetectionFramework:
Erfasst die Tore, sobald der Puck eine Lichtschranke durchquert.

Das MotionCaptureFramework übermittelt den Stream der Kamera. Aus den gewonnenen Bildern wird vom 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.

1.2.2 Java GUI

Die Java GUI wird auf einem Display, welches am Gestell befestigt ist, dargestellt und zeigt den aktuellen Spielstand an. Bislang wurde der Java Code direkt auf dem Raspberry Pi kompiliert. Um

komfortabler entwickeln zu können, kann man nun Änderungen an der GUI am eigenen PC durchführen und nur die exportierte jar Datei auf den Raspberry Pi laden.

1.2.3 Repository

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

1.3 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 Leistung der Lüfter reguliert werden kann. Die Lichtschranken beziehen den Strom vom kleineren Netzteil. Der 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.

Der 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:

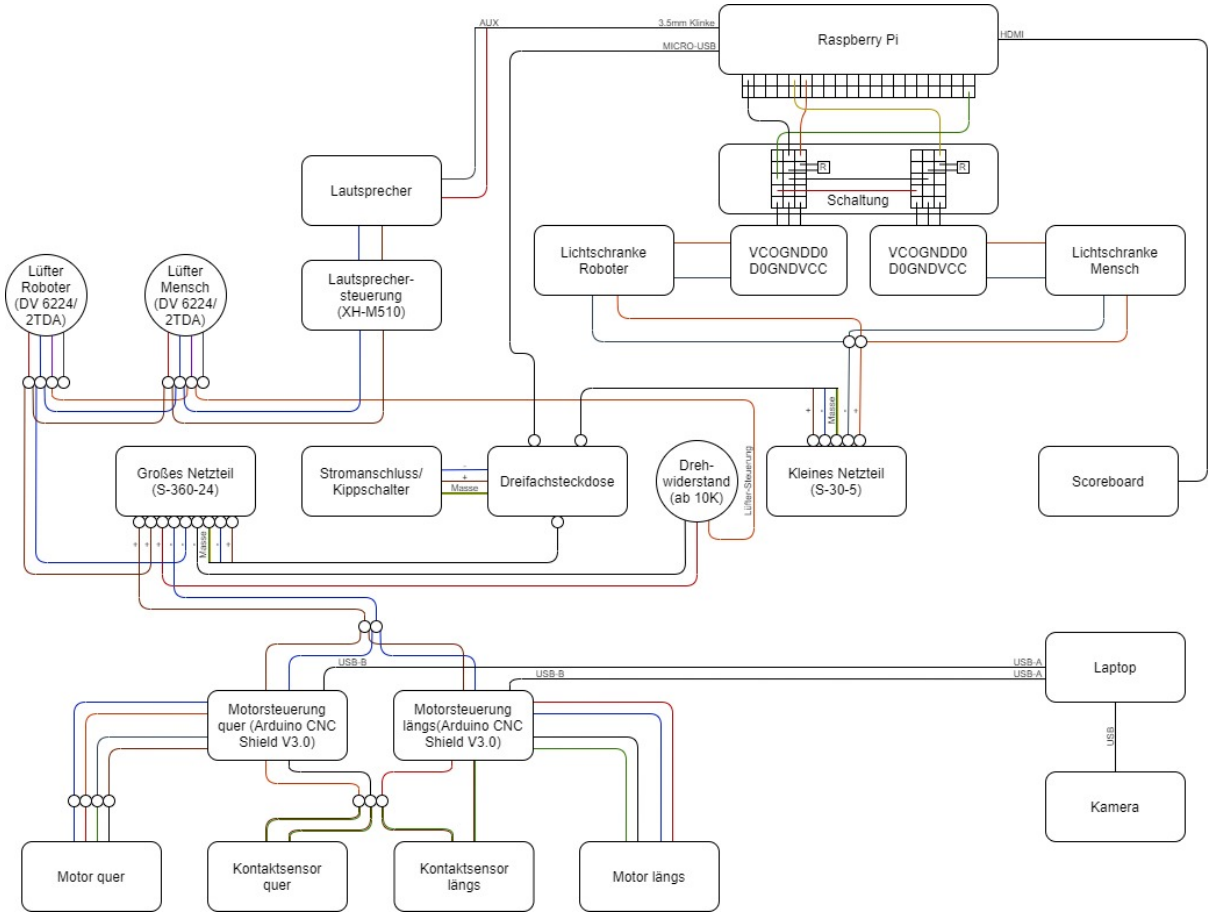

Abbildung 1: Schaltplan (Quelle: Eigene Abbildung)

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). Der Raspberry Pi und das Display sind über 2 Micro-USB-Netzteile angeschlossen.

Lüfter:
Für den Air Flow 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 Leistung der Lüfter gesteuert.

Raspberry Pi:
Der Raspberry Pi führt die Benutzeroberfläche aus und zählt den Punktestand über zwei angeschlossene Lichtschranken.

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

Lichtschranken:
Die Lichtschranken in beiden Toren erkennen, wenn ein Tor geschossen wurde. Sie sind über zwei Module (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:
Beide Achsen des Roboters besitzen einen Kontaktsensor, über den die Motorsteuerung erkennen kann, wann das Ende der Fahrbahn erreicht ist.

Lautsprecher:
Über einen analogen Lautsprecher 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 2: 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.

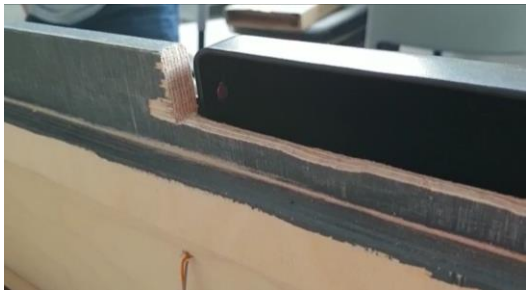

Abbildung 3: Sensor der Lichtschranke (Quelle: Eigene Abbildung)

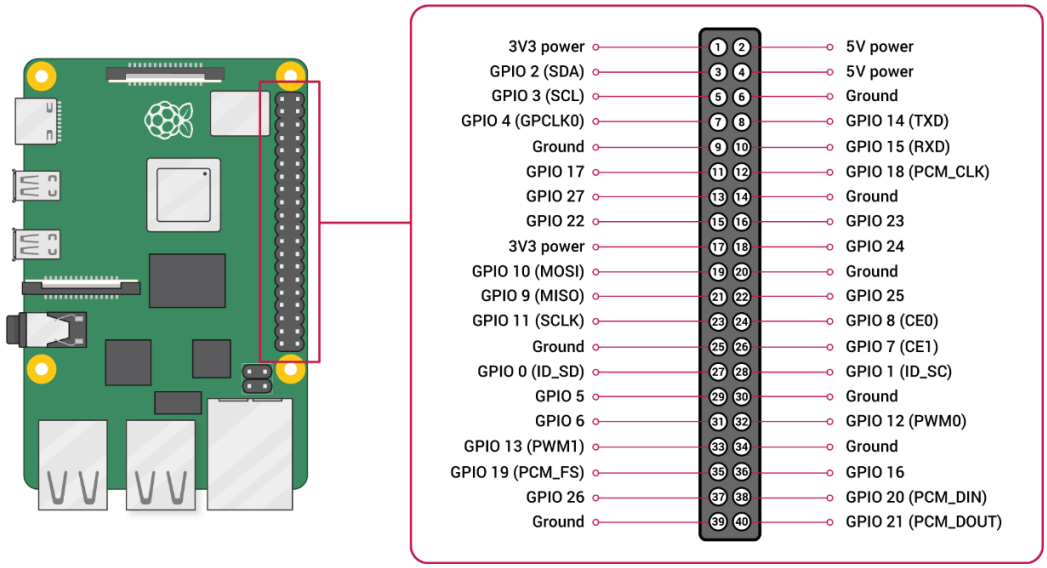

Abbildung 4: Raspberry Pi 3 Pins (Quelle: raspberrypi.org/documentation/usage/gpio, 2020)

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.

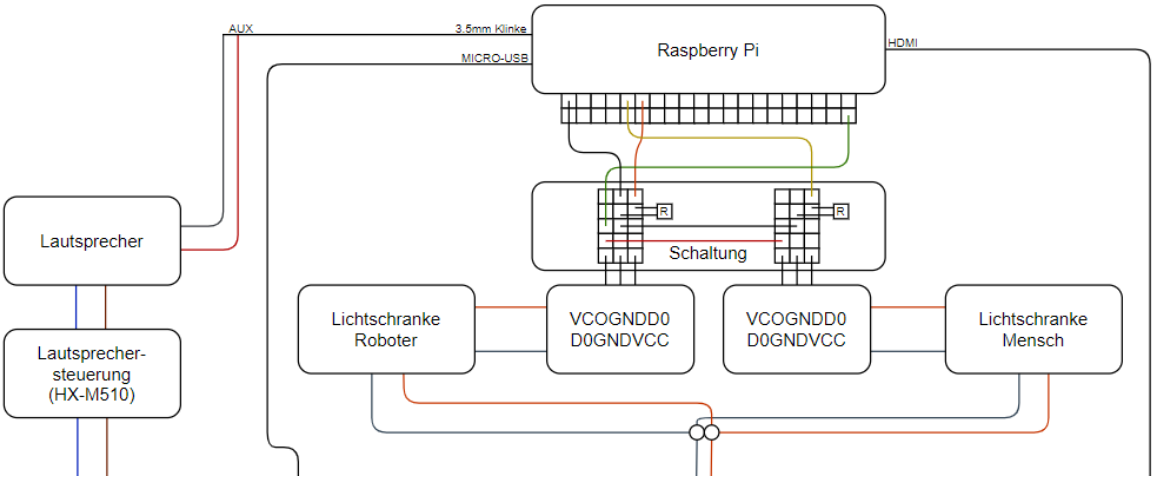

Abbildung 5: Schaltplan der Anschlüsse des Raspberry Pi´s (Quelle: Eigene Abbildung)

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.

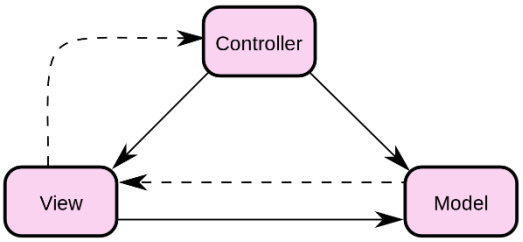

Abbildung 6: MVC Pattern (Quelle: de.wikipedia.org/wiki/Model_View_Controller, 2010)

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 "Air Hockey 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, ein Live-Bild der verwendeten Kamera an, nachdem der Bot gestartet wird. Auf dieselbe Kamera kann aber nicht mehrmals zugegriffen werden. Deshalb wurde bei der Umstrukturierung des Codes der Kamera-ImageProvider, welcher die Bilder für die Puck-

Positionserkennung bereitstellt, erweitert. Er speichert sich dazu das zuletzt erfasste Bild zwischen und stellt dieses dann für das Hauptfenster zur Verfügung.

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.

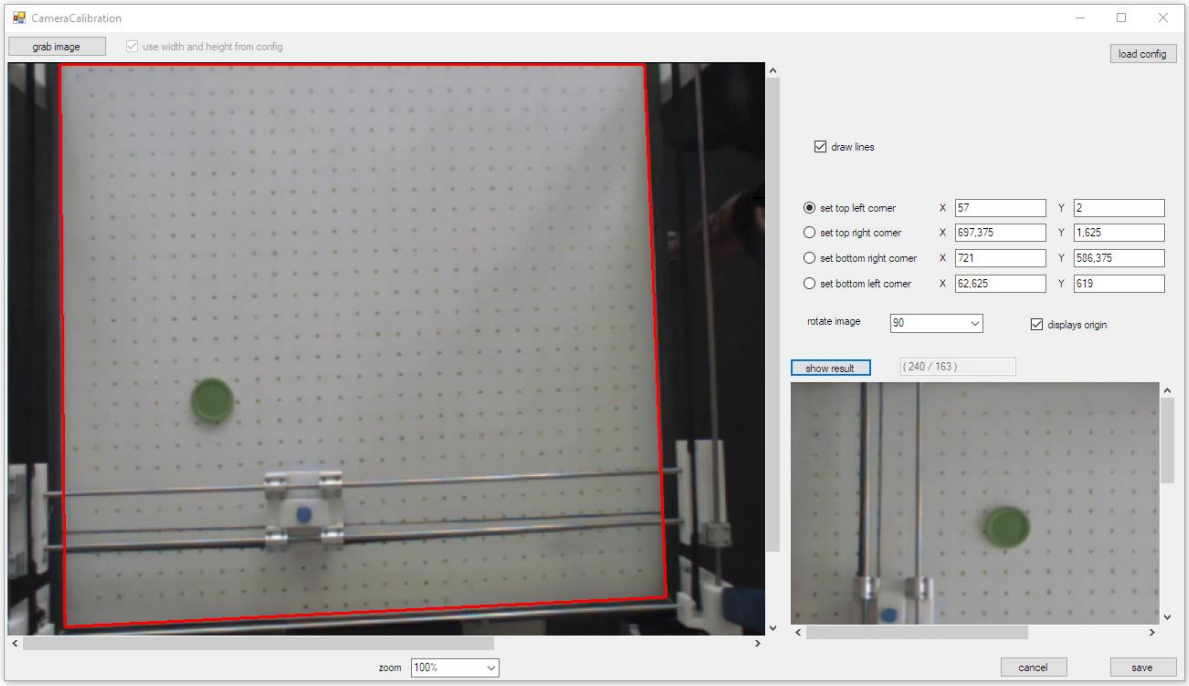

Abbildung 7: Kamera-Kalibrierungsansicht (Quelle: Eigene Abbildung)

In diesem Fenster kann man ein aktuelles Bild der Kamera laden. Im geladenen Bild wird danach markiert, wo das Spielfeld zu sehen ist. Diese Markierung wird mittels vier Punkten vorgenommen, die in das Bild eingezeichnet werden. Um diesen Vorgang zu erleichtern kann das Bild vergrößert werden. Hat man die vier Punkte festgelegt, kann man sich eine Vorschau anzeigen lassen. Bei Bedarf kann man das Ergebnis des Entzerrens / 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 der Bilderkennung, zum Beispiel bei der Kalibrierung der Kamera ausfindig zu machen.

Abbildung 8: ImageDebugging-Fenster

CircleDetectionCalibration-Fenster

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

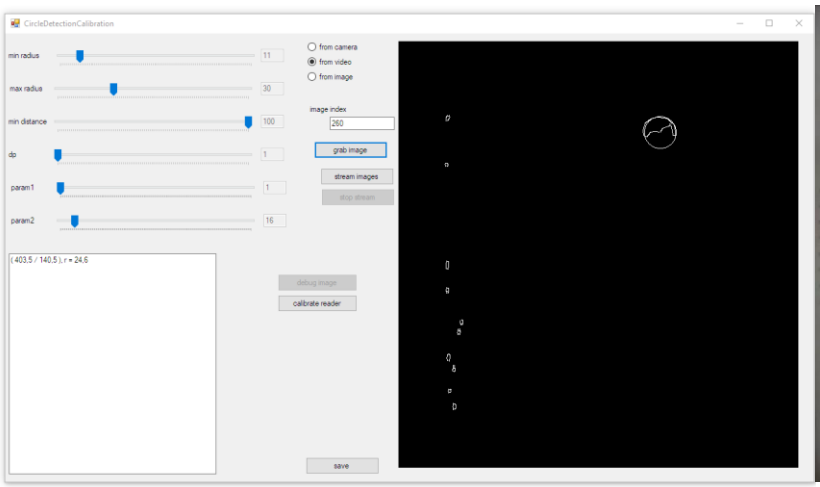

Abbildung 9: Kreiserkennungs-Kalibrierung

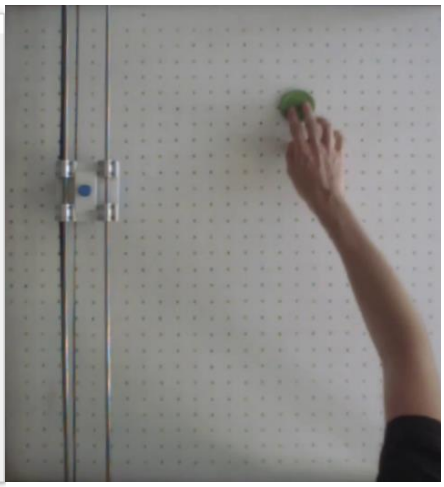

Abbildung 10: Originalbild vor der Verarbeitung

In diesem Fenster kann man die Parameter für die Kreiserkennung 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 noch 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 Algorithmus-spezifische 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.

Pfadvorhersage-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 werden 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 Position trotz Toleranz nicht auf der berechneten Flugbahn befinden, wird die Aktualisierung gestoppt und mittels init() die Vorhersage re-initialisiert.

Implementierung einer Strategien-Komponente
Da der bisherige Code nur darauf aus war die Flugbahn des Pucks zu berechnen und auf die berechnete Flugbahn lediglich mit Blocken 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 SimpleStrategy und der Klasse DirectionForStrategy. Der Ablauf der Strategieberechnung ist folgender:

Zunächst wird eine Position bestimmt, an der der Puck geblockt werden soll, falls keine berechnete Strategie eingesetzt werden kann. Dort wird der Schläger anschließend hinbewegt. Danach wird eine Anzahl Instanzen von Strategieklassen erstellt, momentan vier. Diesen Instanzen wird 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 zurückgegeben, der die Richtung des Pucks zum Zeitpunkt des Schlages angibt und dessen Endpunkt die berechnete Position für die gegebene 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 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 SimpleStrategy Klasse zufällig gewählt. Aus der Richtung des Pucks vor dem Schlag 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 MovementController weiter. Wenn keine Zeit übrig war, eine 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 entspricht in der Größe dem Realen. 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 sich 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.

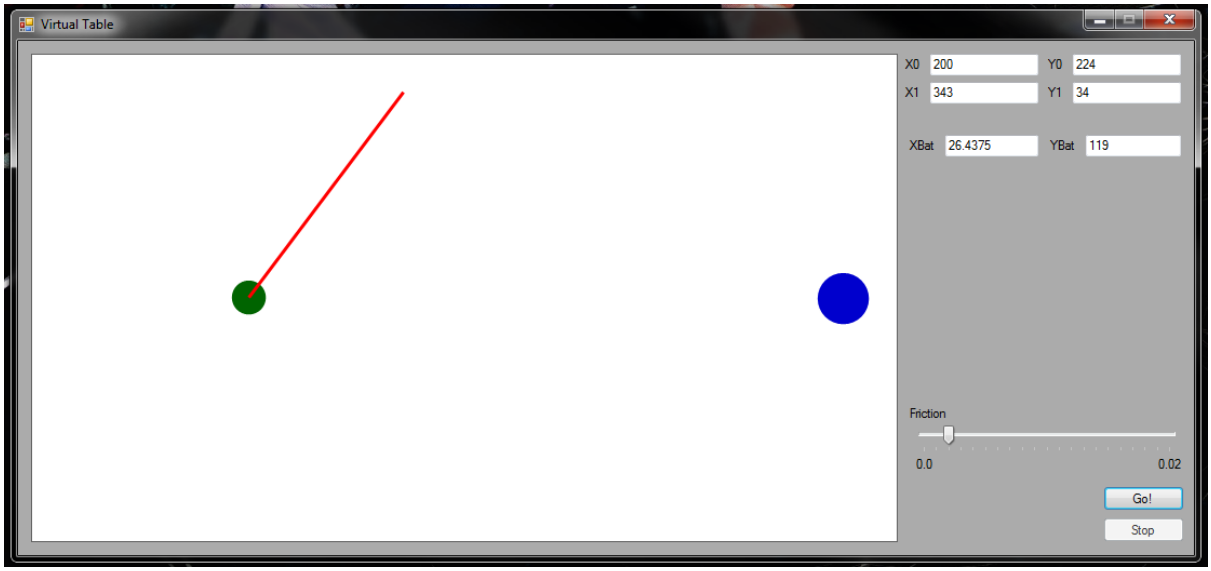

Abbildung 11: Setzen des Geschwindigkeitsvektors

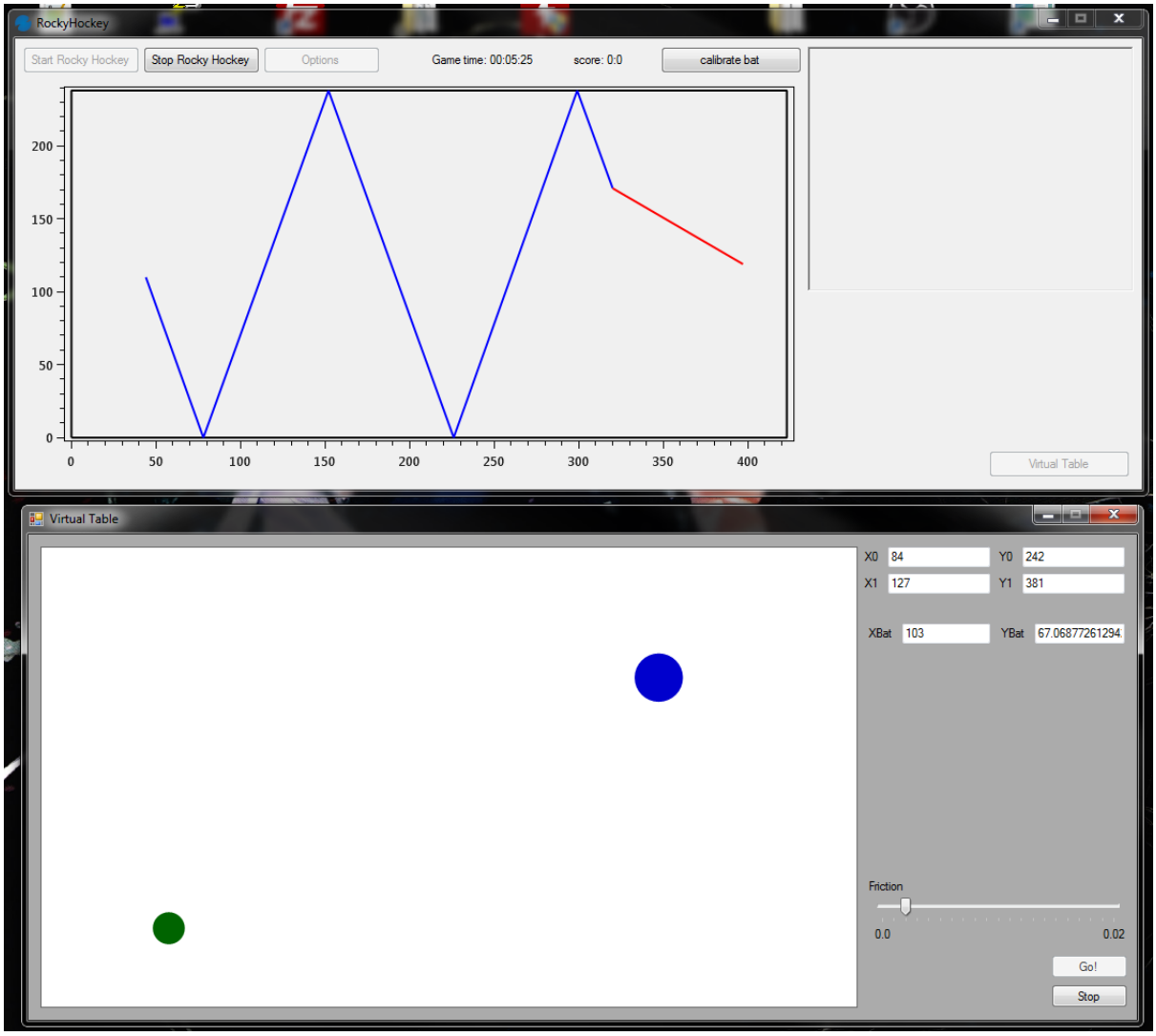

Abbildung 12: Puck in Bewegung; Pfadvorhersage und Schlägerbewegung

2.3 Integration der ArduCam Module

Unter Abschnitt 3 (Hardwareänderungen) ist erklärt, warum wir uns dazu entschieden eine Arducam CMOS Global Shutter AR0134 Kamera in Kombination mit dem ArduCAM USB3.0 Camera Shield zu bestellen. Aus dem Abschnitt geht auch hervor, warum für deren Integration letztlich nur zwei Wochen hatten. In dieser knappen Zeit versuchten wir alles Menschenmögliche um die ArduCam Module zu integrieren:

Unter https://github.com/ArduCAM/ArduCAM_USB_Camera_Shield stellt ArduCam ein GitHub Projekt zur Verfügung, in welchem sich Treiber, Testsoftware und Anleitungen zur Inbetriebnahme der Kamera Module finden lassen. Unter dem Link

http://www.arducam.com/downloads/shields/USB_Shield/ArduCAM_USB_Camera_User_Guide.pdf

ist erklärt, wie die Hardware zusammengesteckt wird, wie die nötigen Treiber auf einem Rechner installiert werden und wie der Demo Code verwendet wird.

Weitere nützliche Dokumentation findet sich unter ArduCAM USB Camera C/C++ SDK/API findet sich unter folgenden Links:

- http://www.arducam.com/downloads/shields/USB_Shield/ArduCAM_USB_Camera_SDK_Guide_V1.3.pdf
- https://www.arducam.com/docs/usb-cameras/software-sdk-and-api/software-sdk-and-api-for-c-c/.

Leider mussten wir feststellen, dass die Dokumentation für unsere Zwecke allenfalls dürftig ist und schon die Verwendung der Demo Software stellte sich als problematisch heraus:

Nach einigem Herumprobieren mit den gegebenen Versionen der USBTest Anwendungen (zu finden im GitProjekt unter Windows -> GUI -> USBTest) konnten wir durch die 32-BitAnwendung die Kamera testweise ansteuern. Dabei stellten wir fest, dass durchaus eine enorme Steigerung der Bildqualität zu sehen ist, die Kamera aber deutlich höher über dem Tisch angebracht werden muss, um diesen vollständig aufzunehmen (insofern die Kamera nicht im Winkel angebracht und kein Bildwarp verwendet wird).

Wir versuchten darauf den Quellcode unter Visual Studio selbst zu kompilieren (zu finden im GitProjekt unter Windows -> CPP -> StreamingDemo), damit er in den bestehenden RockyHockey Frameworks genutzt werden kann. Nach einigen Fehlversuchen konnte wir dann endlich die Software auf dem **Release x64** erfolgreich kompilieren.

In der sich öffnenden GUI wurden keine Konfigurationsdateien angezeigt. Wir füllten daher die für unser Kamera Modell (AR0134) spezifischen Daten manuell aus, welche sich aus den Demo Anwendungen auslesen lassen, da dort alle Konfigurationsdateien angezeigt werden und die richtige, wenn die Module über USB angeschlossen sind, auch automatisch ausgewählt wird. Leider traten dann beim Klicken auf “Play” durchgehend Fehler auf, für die es keine Dokumentation von ArduCam gibt. Nach einiger Zeit hatten wir aber dennoch endlich ein Bild in einem Visual Studio Projekt, man muss dazu jedoch die Demo Anwendung zuerst starten und dann das Visual Studio Projekt. Der Stream wird dann in beiden Fenstern angezeigt. Durch die

späte Lieferung war uns dann aber klar, dass wir die Integration bis zur Projektpräsentation nicht würden abschließen können. Daher entschieden wir uns, dies dem nächsten Projektteam als großen Meilenstein zu überlassen und die letzten Tage noch mit der PS3 Kamera zu arbeiten.

Parallel versuchten wir durchgehend mit dem ArduCam Support Team Kontakt (unter anderem unter https://www.arducam.com/service/) aufzunehmen, um die auftretenden Fehler klären zu können. Wir erhielten jedoch keine Antwort mehr, ArduCam schien während der Corona Zeit nicht genügen Kapazitäten zu haben. Wir hoffen, das nächste Projektteam hat hier mehr Glück.

3. Hardwareänderungen

3.1 Hebefunktion der Tischplatte

Eine Änderung, die vorgenommen wurde, ist eine Vorrichtung, mit welcher sich die schwere Platte leichter vom Tisch entfernen lässt. Davor musste man diese mit viel Aufwand durch Hilfsmittel wie Schraubenschlüssel hochhebeln, um sie dann greifen und hochziehen zu können. Allein war diese Arbeit nochmal schwieriger und es bestand ein hohes Risiko sich die Finger einzuklemmen. Da die Platte jedes Mal entfernt werden musste, wenn Änderungen am Tisch vorgenommen oder an der Elektronik gearbeitet werden wollte und man durch die Covid-19-Pandemie oft alleine am Tisch arbeitete, wurde beschlossen eine Vorrichtung zu bauen, die helfen sollte die Tischplatte einfach alleine entfernen zu können.

Die Funktionsweise ist wie folgt: unter die Platte wurden zwei Holzteile geschraubt, an denen jeweils ein Faden befestigt ist. Diese Fäden führen an den Ecken des Tischs nach oben und können um zwei kleine Schrauben gewickelt werden, um nicht im Weg zu sein. Da die Fäden direkt in den Ecken des Tisches verlaufen, können sie die Bahn des runden Pucks auch nicht stören und haben so keinen Einfluss auf das Gameplay. An diesen beiden Fäden kann die Platte mit je einer Hand hochgehoben werden. Die Fäden wurden relativ dick ausgewählt, um nicht unnötig zu schmerzen oder sogar in die Haut zu schneiden. Der zweite Teil der Vorrichtung umfasst zwei „Schnapper“ welche die Platte fixieren, sobald sie hoch genug über dem Tisch sind. Sobald die Platte fixiert ist, kann man die Fäden loslassen und die Platte ohne Probleme vom Tisch ziehen.

Damit man die Platte auch wieder problemlos allein in den Tisch lassen kann, haben die zwei Schnapper je ein Loch. Durch diese Löcher und ein Luftloch der Platte muss man je einen kleinen Pin (in unserem Fall Metallstäbchen) schieben und man kann die Platte an den beiden Schnüren wieder in dem Tisch sinken lassen. Danach werden die Pins wieder herausgezogen und die Schnapper sind wieder funktionsbereit, sodass der Mechanismus wiederverwendet werden kann.

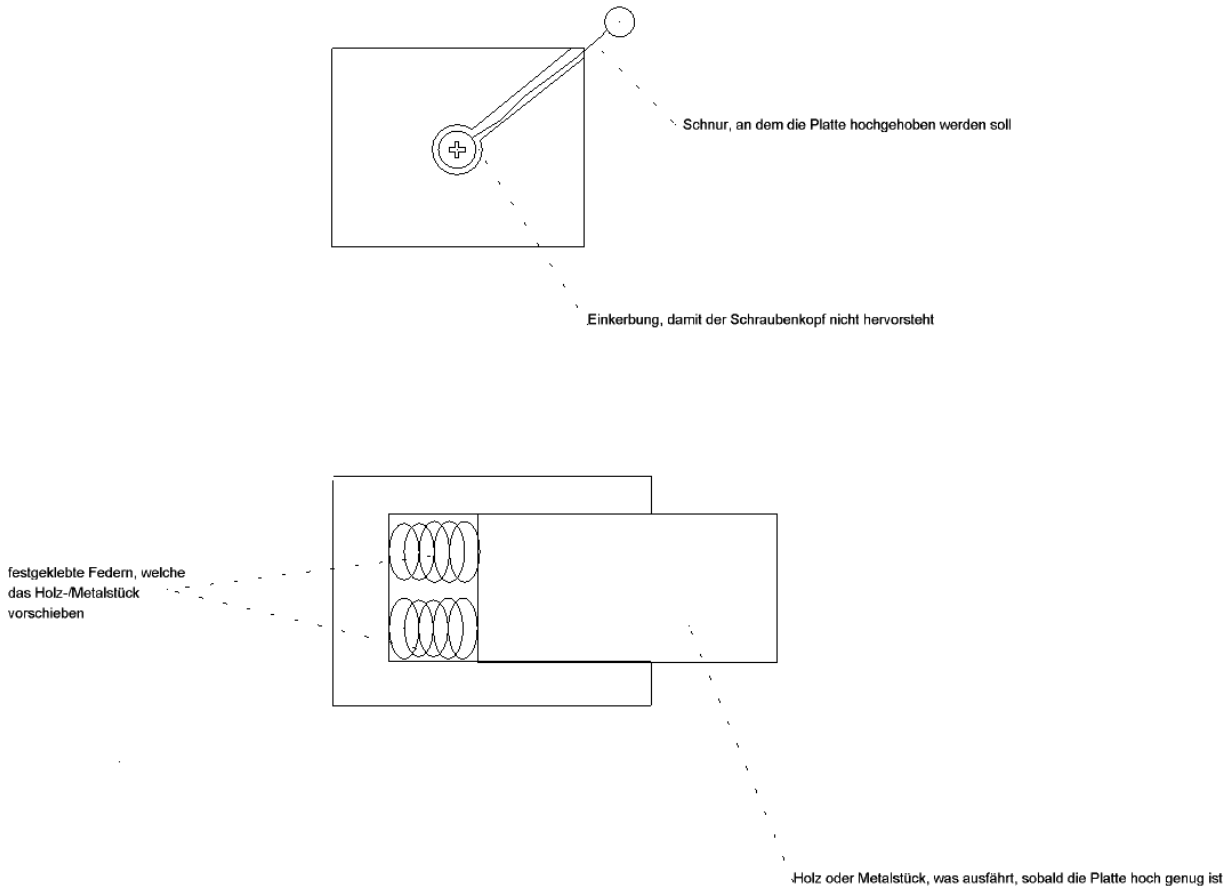

Abbildung 13: Mechanismus für das Heben der Platte (Quelle: Eigene Abbildung)

3.2 Neues Kameragestell

Abbildung 14: Kameragestell vorher (Quelle: Eigene Abbildung)

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, ein Neues zu bauen.

Nachher:

Die neue Kamerahaltung passt gut in die unten vorgesehenen Halterungen, da 4cm x 6cm 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 15: Kameragestell nachher 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 16: Kamerabefestigung (Quelle: Eigene Abbildung)

Abbildung 17: Kameragestell nachher vorne (Quelle: Eigene Abbildung)

3.3 ArduCam Module

Bereits das zweite und dritte Projektteam musste feststellen, dass die verwendete PS3-Kamera diverse Probleme durch die schlechte Treiberunterstützung und miserable Auflösung verursacht. Die geringe Auflösung von nur 320x240px hat zur Folge, dass der Puck um bis zu 4px flattert und die Aufnahme auf den halben Tisch beschränkt ist, was bedeutende Abweichungen in der Flugbahnberechnung zur Folge hat. Daher haben wir uns dazu entschlossen eine neue Kamera über die Hochschule zu bestellen. Nach längerem Abwägen und verschiedenen Empfehlungen in online Foren entschieden wir uns für die Arducam CMOS Global Shutter AR0134 in Kombination mit dem ArduCAM USB3.0 Camera Shield. Mit dieser ist es möglich 54 Frames pro Sekunde bei maximaler Auflösung (1280x960) aufzunehmen und entsprechend höhere Framerates bei niedrigeren Auflösungen. Damit ist es möglich den gesamten Tisch aufzunehmen und dabei gleichzeitig das Pixelflattern einzuschränken. Auch versprachen die Beschreibungen eine einfache Integration und leichte Erweiterung der Module, dies bestätigte sich später jedoch leider nicht.

Abbildung 18: ArduCam AR0134 mit USB Camera Shield (Quelle: https://www.arducam.com/product/arducam-cmos-ar0134-1-3-inch-1-2mp-monochrome-camera-module/)

Die Bestellung erfolgte über Professor Deinzer und die Hochschule. Obwohl wir die Bestellung bereits sehr früh in die Wege geleitet hatten, mussten wir, bedingt durch die Hochschul-Bürokratie und die Coronasituation, ärgerlicherweise bis zwei Wochen vor unserer Abschlusspräsentation warten, bis wir die Kameramodule endlich in Händen hielten. Dies ließ nur sehr wenig Zeit für die Integration der Module, worauf in Kapitel 2.3 schon genauer eingegangen wurde.

3.4 Elektronik und Verkabelung

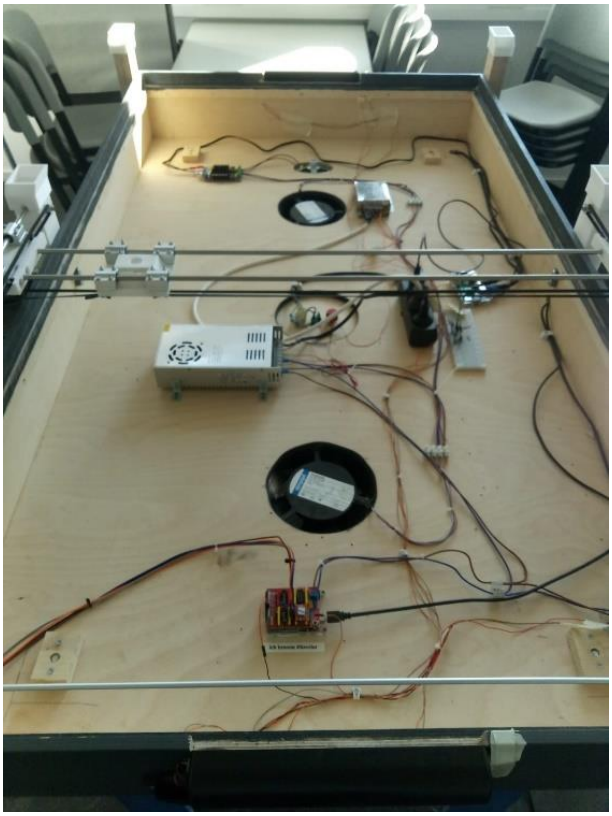

Abbildung 19: Tischverkabelung zuvor (Quelle: Eigene Abbildung)

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.

Nachher:
Zunächst wurde der Schaltplan erstellt, der bereits im Ist-Zustand der Hardware 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 Lichtschranken) 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.

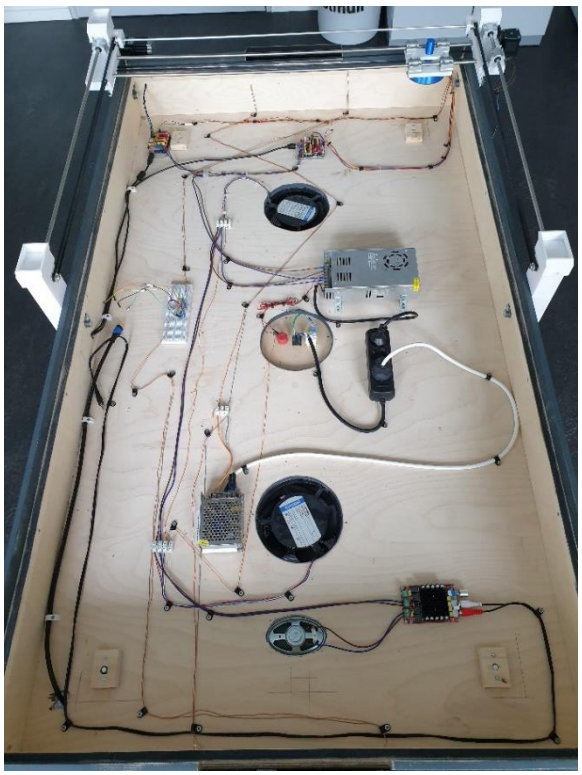

Abbildung 20: Tischverkabelung danach (Quelle: Eigene Abbildung)

3.5 Aufbau der Zwischenplatte

Bei den ersten Tests des Tisches fiel auf, dass sich die Luft durch den Tisch nicht gleichmäßig verteilt, was zu einem ungleichmäßigen 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 ausreichend versorgt. Als erstes wurde überlegt, die Lüfter höher zu befestigen. Am Ende entschied man sich jedoch eine weitere Zwischenplatte einzubauen.

Abbildung 21: Zwischenplatte am Tisch angebracht (Quelle: Eigene Abbildung)

Diese besteht aus zwei Löchern an den Positionen 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.

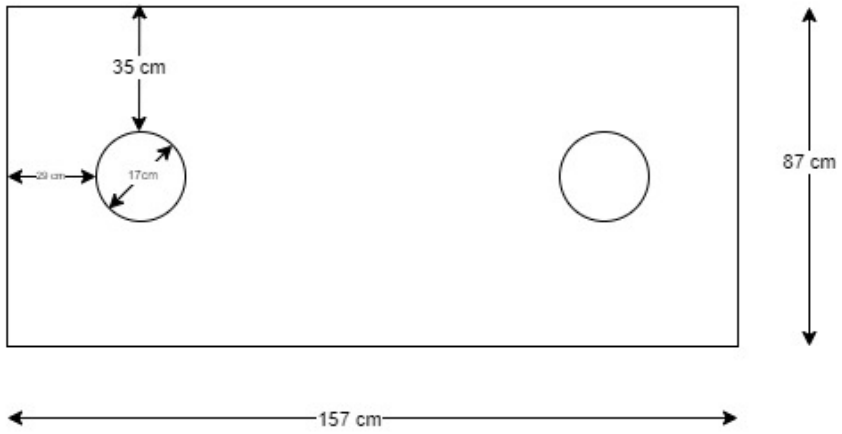

Abbildung 22: Maße/Aufbau der Zwischenplatte (Quelle: Eigene Abbildung)

4. Probleme

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 und Fragen beantwortet haben. Wir wurden teilweise mit offenen Fragen zurückgelassen und der Information, wir sollen uns offene Fragen aus dem Abschlussbericht erarbeiten. 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.

Arbeiten am Tisch

Durch die COVID-19 Pandemie durfte man anfangs höchstens zu zweit am Tisch arbeiten, was sich schon beim erstmaligen Aufstellen des Tisches als große Herausforderung herausstellte. Der Tisch passt nicht durch eine normale Türe, sodass er gekippt durch diese gebracht werden musste. Anfangs war es auch schwierig die Platte vom Tisch zu heben, um an den Raspberry Pi oder andere elektronische Komponenten zu gelangen. Dieses Problem wurde dann durch einen selbstgebauten Mechanismus (siehe 3.1) behoben.

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.

Bestellung und Integration der Kameramodule

Die Probleme die bei der Bestellung und Integration der Kameramodule auftraten wurden schon oben beschrieben. Die zwei Hauptursachen für das Auftreten der Probleme waren die Corona Pandmie und der große Zeitmangel. Nur 2 Wochen, während welchen zudem noch die Prüfungsphase war oder zumindest direkt anstand, reichten einfach nicht aus, um sich genügendmit der Dokumentation der Kameramodule zu beschäftigen und diese zu integrieren. Gleichzeitig stellten sich die ArduCam Module als eher schwer handhabbar heraus. Es war nicht möglich, diese einfach als generische Kamera von einem Rechner erkennen zu lassen und direkt in unser Projekt einzubinden. Stattdessen müssen die ArduCam eigenen Treiber und Bibiliotheken verwendet werden. Zusätzlich gibt es keine Codebausteine für C#. Es steht jedoch eine .dll Datei zur Verfügung, welche auch im C# Projekt eingebunden werden kann. Der ArduCam Support ließ sich in der kurzen Zeit nicht kontaktieren.

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.

Elektronik 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 entweder verlötet werden oder durch durchgehende Kabel ersetzt werden.

ArduCam Module

Befolgt die Schritte und schaut euch die Links an, die hier im Bericht zu den ArduCam Modulen stehen, damit sollte euch eine steile Lernkurve bevorstehen. Wenn ihr es schafft, die Module ordentlich in das Projekt einzubinden solltet ihr einen echten Sprung in der Performance des Roboters erreichen können. Um mit der Kamera den ganzen Tisch senkrecht aufnehmen zu können, müsst ihr sie sehr weit oben aufhängen, wir haben sie direkt an die Decke gehangen. Alternativ könnt ihr sie natürlich auch im Winkel auf den Tisch blicken lassen und das Bild mit EmguCV warpen.

6. Anleitungen und Tipps

Ausarbeitungen

Dokumentationen, 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.

Elektrotechniker

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

3D Druck

Viele Teile des Tisches wurden über 3D Druck erstellt. Immer, wenn teile verbessert oder auf Grund von verschleiß ausgetauscht werden sollten, brauchten also auch wir einen 3D Drucker. Den Vorgängerteams standen eigende 3D Drucker oder 3D Drucker aus dem FabLab Allgäu zur Verfügung, welches jedoch zu unserem Projektstart nicht mehr existierte. In unserem Team besaß auch niemand einen 3D Drucker. Nach langem Herumgefrage kamen wir letztendlich an die Kontakte der Technikerwerkstatt der Hochschule und einen weiteren privaten Kontakt. Von beiden Seite erhielten wir die Erlaubnis, einzelne Teile dort drucken zu dürfen. Dies taten wir auch prompt, um zum Beispiel den kaputten Schläger des Roboter austauschen zu können (die STL Files finden sich in unserem GitHub Projekt im „wiki-resources“ Branch). Wir hinterlassen hier die Kontakte für das nächste Projektteam:

- 3D Druck Technikerwerkstatt: beck.joachim@web.de
- 3D Druck Privat/Klevertec: oliver.hauser@hs-kempten.de>

Bestellungen

Die Mühlen an der Hochschule mahlen sehr langsam, vermutlich auch außerhalb der Corona Zeiten. Wenn ihr neue Teile bestellen wollt, macht das direkt am Anfang! Teilt Professor Deinzer so schnell wie möglich eure Pläne mit, damit er diese prüfen und euren Antrag an die richtige Stelle weiterleiten kann.