# Dokumentation RockyHockey

# Wichtige Kontakte

| Name | Bereich | Kontakt/Alternative |
|---|---|---|
| Prof. Dr. Stefan Rieck | Admin Open Project Server | stefan.rieck@hs-kempten.de |
| Prof. Dr. Arnulf Deinzer | Projektleitung | arnulf.deinzer@hs-kempten.de |
| Dr. Dietmar Prestel | Projektleitung | dietmar.prestel@hs-kempten.de |
| Hannes Mücklich | Teamleitung, Algorithmen | hannes.muecklich@stud.hs-kempten.de<br>hannesmuecklich@t-online.de |
| Tim Brugger | Admin Bitbucket, Bilderkennung | tim.brugger@stud.hs-kempten.de<br>timjablonka@hotmail.de |
| Mathias Wilhelm | Pucktracking, Oberfläche | mathias.wilhelm@stud.hs-kempten.de<br>mathias.wilhelm@tele2.at |
| Matthias Kiechle | Hardware, Motoransteuerung | matthias.t.kiechle@stud.hs-kempten.de<br>mtk_matthias@yahoo.de |

# Wissensdatenbank

Alle wichtigen Daten findet man im Projekt “Rocky Hockey”, welches im Open Project Server der Hochschule liegt. Erreichbar ist die Seite im Hochschulnetz unter
https://ops.hs-kempten.de/redmine/projects/rocky-hockey
Unter den Kategorien “Wiki” und “Dateien” finden sich nützliche Dateien und Wissensammlungen über diverse Themen.
Um einen Zugang zu erhalten, muss man sich mit Herrn Rieck in Verbindung setzen.

# VCS

Als Versionsverwaltung wurde Bitbucket genutzt. Erreichbar ist das Projekt unter
https://bitbucket.org/rockey_hockey/
Um einen Zugang zu erhalten, bitte mit Tim Brugger in Kontakt treten.

# EmguCV auf einem Raspberry Pi installieren

Um die EmguCV-Bibliothek auf einem Raspberry Pi zu installieren muss zunächst mono mit dem Befehl

```
sudo apt-get install mono-complete
```

installiert werden. Mono wird dient zum Ausführen von C#-Code unter Linux. Danach muss der EmguCV-Code auf dem Raspberry mithilfe von

```
git clone https://github.com/emgucv/emgucv emgucv
```

ausgecheckt werden. Mit

```
cd emgucv
```

kommt man in das ausgecheckte EmguCV-Verzeichnis, um dort die Untermodule des Repositories zu klonen. Dafür muss man den Befehl

```
git submodule update --init -recursive
```

auf der Konsole ausführen.
Als nächstes muss das Verzeichnis gewechselt werden. Hierfür wird das Kommando

```
cd platforms/raspbian
```

verwendet, da ja die Raspberry-Version von EmguCV installiert werden soll. Anschließend müssen die Abhängigkeiten für EmguCV durch das Ausführen von

```
./apt_install_dependency
```

installiert werden. Das installieren von EmguCV auf einem Raspberry Pi dauert ca. sechs Stunden. Mit dem Befehl

```
./cmake_configure
```

konfiguriert sich EmguCV selbst.
Wurden all diese Schritte erfolgreich durchgeführt, dann kann man im Verzeichnis „libs“ im ausgecheckten EmguCV-Ordner die notwendigen dll-Dateien finden, die für die Ausführung eines EmguCV-Programmes benötigt werden. Um ein Release auf dem Raspberry zu deployen, müssen alle Dateien aus dem libs Ordner mit der Endung „.dll“, „.a“, sowie die Datei“ mit dem Namen „cvextern“ in das Release-Verzeichnis des C#-Projektes kopiert werden. Zusätzlich sollte man die EmguCV-dlls, die bei einer Kompilierung unter Windows erzeugt werden, aus dem Release Ordner gelöscht werden. Diese Dateien befinden sich in dem x64- und dem x86-Ordnern innerhalb des Release-Ordners.

# Ermittlung von Vektoren mittels fünf Bildern

In der Spielschleife von RockyHockey werden bei jedem Durchlauf zu Beginn solange Bilder aufgenommen, bis auf insgesamt fünf Bildern der Puck erkannt wurde. Aus diesen fünf Punkten versucht der Code einen Mittelwert zu finden, um die Flugbahn des Pucks möglichst genau zu bestimmen.
Der Vektor, der die Flugbahn des Pucks beschreibt, hat den letzten der fünf erkannten Punkte als Startposition. Die Richtung des Pucks ergibt sich aus der berechneten Steigung, wobei hier insgesamt die durchschnittliche Steigung zwischen den Punkten berechnet wird. Ist unter diesen Steigungen ein Wert zu finden, der zu stark zur davor berechneten Steigung ist, dann wird dieser Wert nicht weiter berücksichtigt.
Um die Geschwindigkeit des Pucks zu berechnen, wird zunächst die Länge zwischen zwei hintereinander ermittelten Punkten in Pixel berechnet. Zusätzlich merkt sich das Programm, wie viele Frames zwischen zwei erkannten Punkten liegen, wodurch auch die Zeit für die zuvor berechnete Strecke gegeben ist.
Die Zeit für die Länge eines Vektors ergibt sich also aus dem Abstand der Frames geteilt durch die Anzahl der aufgenommenen Frames pro Sekunde. Um nun die Geschwindigkeit in Pixel pro Sekunde zu erhalten muss die Länge des Vektors durch die berechnete Zeit geteilt werden.

# Implementierung der Spielschleife

Beim Start von RockyHockey werden zunächst alle notwendigen Module initialisiert und anschließend die Spielschleife gestartet. Die Spielschleife ist als eine Endlosschleife implementiert, die asynchron in einem eigenen Thread gestartet wird. In der Spielschleife werden zu Beginn solange Bilder aufgenommen, bis der Puck auf fünf erkannt wurde. Aus diesen fünf Punkten wird der Vektor, der die Flugrichtung und die Geschwindigkeit des Pucks beschreibt, berechnet. Anschließend wird der Schlag berechnet, welcher sich aus der Flugbahn und der Geschwindigkeit des Pucks ergibt. Zum Schluss wird die Schlagbewegung an die Motorensteuerung weitergeleitet, damit diese die Bewegung ausführen können.

# Berechnung der Flugbahn

Um die Flugbahn bis zu einer festgelegten geraden zu erhalten, muss der Vektor des Pucks verlängert werden. Trifft der Vektor dabei auf die Spielfeldgrenzen, dann muss er dort reflektiert werden. Hierbei ist anzunehmen, dass der Einfallswinkel gleich dem Ausfallswinkel ist. Hat man alle Vektoren verlängert erhält man die Position, an welcher der Puck beim Roboter ankommt.

# Berechnung Schläger-Kreisbahn

Die Berechnung einer Kreisförmigen Bewegung des Schlägers dient dazu, dass der Puck aus jeder Position auf das gegnerische Tor geschossen werden kann. Dies funktioniert nicht nur bei direkten Schlägen, sondern auch für Schläge über eine oder sogar mehrere Banden. Dadurch muss für verschiedene Strategien nur die gewünschte Flugbahn des Pucks berechnet werden, die Berechnung danach ist immer gleich. Die Flugbahn des Pucks ist eine Tangente zu dem abgefahrenen Kreis.
Die Berechnung des Kreises auf dem die Bewegung liegt ist relativ komplex. Es sind nur zwei Punkte auf dem Kreis bekannt, nämlich der des Schlägers und der Punkt an dem der Puck gespielt wird. Zusätzlich lässt sich aber noch die Senkrechte auf die gewünschte Flugbahn in dem Punkt an dem der Puck geschlagen wird berechnen. Mit diesen Informationen lässt sich der Kreismittelpunkt berechnen. Mit der Kreisformel  lassen sich für die zwei bekannten Punkte zwei Formeln mit  aufstellen. Da beide Punkte auf demselben Kreis liegen und somit denselben Radius haben können die Formeln gleich gesetzt werden. Nun kann nach der y-Position des Kreismittelpunktes  aufgelöst werden. Da die vorher bestimmte Senkrechte durch den Mittelpunkt des Kreises geht sind die y-Positionen der beiden Formeln an einem Punkt gleich. Durch gleichsetzten ist nur noch die x-Position des Kreismittelpunktes unbekannt. Dieser kann nun nach auflösen der Formel berechnet werden. Anschließend kann die y-Position des Kreismittelpunktes mit der Formel der Senkrechten bestimmt werden. Mit dem Mittelpunkt kann nun auch der Radius berechnet werden.
Mit der oben genannten Kreisformel können nun weitere Punkte auf dem Kreis berechnet werden. Für den Schläger sind lange Wege schneller zu fahren, da dieser eine Beschleunigung hat. Aus diesem Grund muss ein Mittelweg zwischen einer annähernden Kreisbewegung und der Geschwindigkeit des Schlägers gefunden werden. Bisher wurden fünf Vektoren auf dem Kreis berechnet. Bei der Berechnung der Punkte auf dem Kreis wird eine feste x-Position gesetzt. Am einfachsten geschieht dies durch ein gleichmäßiges aufteilen des Abstandes zwischen Schlägerposition und der Position an der der Puck geschlagen wird. Anschließend kann die passende y-Position berechnet werden. Bei der Berechnung der Punkte muss darauf geachtet werden was für eine Hälfte des Kreises berechnet wird. Bei der Berechnung sollte außerdem noch darauf geachtet werden über was für eine Bande der Puck geschlagen werden soll und in welcher y-Hälfte der Punkt ist an dem der Puck geschlagen wird. Zusätzlich darf die Kreisbahn natürlich auch nicht außerhalb des Spielfeldes liegen.

Bei der Berechnung wird in seltenen Fällen noch die falsche Kreishälfte berechnet. Damit die kreisförmige Bewegung wirklich Sinn macht muss der Schläger gleichzeitig mit dem Puck an der Schlagposition sein. Deswegen muss eine Wartezeit vor der Bewegung für den Schläger berechnet werden. Dies verursachte aufgrund von Rundungsfehlern an vielen Stellen Fehler. Ein weiteres Problem ist das warten zwischen den einzelnen Vektoren. Der Schläger darf immer nur eine Position an die er fahren darf erhalten. Deswegen muss zwischen den einzelnen Vektoren gewartet werden. Diese Wartezeiten waren teilweise viel zu lang.
Implementiert ist außerdem schon eine mögliche Beschleunigung innerhalb der Kreisbewegung. Hierfür muss jedem Vektor die gewünschte Geschwindigkeit des Schlägers mitgegeben werden. Momentan ist dies die maximale Geschwindigkeit. Um dieses Feature tatsächlich umsetzten zu können müsste die Motorenansteuerung komplett überarbeitet werden.
Für neue Taktiken für die Flugbahn des Pucks muss wie bereits angesprochen lediglich eine andere Tangente für den Kreis berechnet werden. Dies geschieht in eigenen Klassen und ist somit leicht erweiterbar. Per Zufall oder auch nach System kann dann bei jedem Schlag ausgewählt werden welche Taktik benutzt werden soll.
Um die Kreisbewegung einzusetzen muss nur der direkte Schlag aus kommentiert und der Rest ein kommentiert werden.

# LED-Beleuchtung

Die LEDs werden durch einen Arduino gesteuert. Dieser erhält seine Kommandos über den Raspberry Pi über USB. Um dem Arduino Kommandos schicken zu können muss auf dem Raspberry Pi ein Serieller Port geöffnet werden. Dies geschieht unter Linux über spezielle Dateien. Wenn der Port geöffnet ist reicht es über ein „Serial.Write“ Kommando mit Methodennamen auf dem Arduino die gewünschte Animation aufzurufen. Neue Animationen können durch das Hinzufügen von Methoden und der Abfrage im mainloop auf dem Arduino einfach ergänzt werden. Zu beachten ist, dass die LEDs sehr viel Strom brauchen. Bei komplett weißer Beleuchtung ist ein Stromverbrauch von über 7 Ampere erreicht worden. Das Netzteil schafft allerdings nur 5 Ampere. Aus diesem Grund wird nun bei einem weißen Licht nur jede zweite Lampe beleuchtet. Bei einer bunten Beleuchtung spielt das nach den jetzigen Erfahrungen keine Rolle mehr. Die Lötstellen zwischen den einzelnen Streifen sollten nicht stark bewegt werden, diese könnten sonst kaputt gehen. Die LED Streifen sollten außerdem nicht betrieben werden wenn das Netzteil aus ist, da diese ansonsten entweder über das Raspberry Pi oder über den angeschlossenen Laptop betrieben werden.

# Touchscreen

Die originalen Touchtreiber hatten die Probleme, dass der Touchscreen am Rand nicht reagiert hat und der Touch recht ungenau war. Bessere Erfahrungen wurden mit den Treibern von der Seite
https://github.com/CytronTechnologies/xpt2046-LCD-Driver-for-Raspberry-Pi

gemacht. Hat man die Treiber installiert, müssen noch die Parameter für die Rotation des Bildschirms sowie die Invertierung der X- und Y-Achse gesetzt werden.
Dies geschieht in folgenden Dateien

```
/usr/share/X11/xorg.conf.d/10-evdev.conf
/usr/share/X11/xorg.conf.d/45-evdev.conf
/boot/config.txt
```

Bitte als Vorlage die bestehenden Dateien auf der SD-Karte im Raspberry nutzen.

# Kameratreiber

Die Playstation 3 Kameras unter Windows zu betreiben, gestaltete sich mitunter recht schwierig. Es gibt wohl diverse Möglichkeiten, die Kameras direkt seriell anzusprechen, was wir aber aufgrund des Aufwandes nicht weiter verfolgten. Nach langer Recherche haben wir Treiber von
https://codelaboratories.com/home/
gefunden, welche im Open Project Server unter Dateien abgelegt wurde. Damit war es möglich, eine Kamera aus dem Programm heraus ohne Weiteres anzusteuern. Auch waren damit bis zu 187 fps möglich, was sehr zur Geschwindigkeit der Kameraerkennung beigetragen hat.
Leider ist es nicht ohne Weiteres möglich, eine zweite Kamera anzusteuern. Dies soll zwar mit den o.g. Anbieter möglich sein, jedoch ist die Seite seit geraumer Zeit nicht mehr vom Entwickler gepflegt worden und die Dokumentation ist sehr spärlich.
Unter Linux sollte laut Vorgängerteam eine Ansteuerung beider Kameras recht einfach mit oboard-Treibern möglich sein. Zu beachten ist, dass es wohl einen Parameter geben soll, welcher eine höhere Anzahl von fps ermöglichen soll.

# Kameraerkennung

Die Kameraerkennung erfolgt mittels einer Playstation 3 Kamera mit 320x240px und einer Framerate (unter Windows) von 187 fps. Als Framework zur Kameraerkennung wird EmguCV genutzt, welcher ein Wrapper für C# für das Framwork OpenCV darstellt.

Für OpenCV ist die hat die linke, obere Ecke die Position (0|0). die X-Achse beschreibt hierbei die Richtung nach rechts, die Y-Achse die Richtung nach unten. Auf diesen Umstand muss man besonders achten, wenn man zwei Kamerabilder zusammensetzen möchte und wenn man die Koordinaten an das Bewegungs-Tracking-Framwork weitergeben möchte, da da das Koordinatensystem sowie Nullpunkt am Tisch spezifisch festgelegt wurde. Die Spezifikation hierzu kann man dem Pflichtenheft entehmen.

Der erste Schritt, nachdem ein Bild aufgenommen wurde ist das “Warpen” und Ausschneiden des Bildes. Damit die Kamera das Spielfeld aufnehmen kann, ist eine leichte Neigung von Nöten. Das heißt, die Kamera zeigt nicht senkrecht auf das Spielfeld. Resultat ist, dass die Kamera einen “Trapezeffekt” hervorruft, was zur Folge hat, dass das Spielfeld nicht mehr rechteckig und die Banden nicht mehr parallel sind.

Um den zu begegnen, kann man mithilfe der Kommandos

```
CvInvoke.GetPerspectiveTransform(PointF[] src, PointF[] dest);
CvInvoke.WarpPerspective(Mat sourceImage, Mat resultImage, Mat lambda,
Size new Size(int, height));
```

das Bild warpen und das Spielfeld wieder rechteckig darstellen. Die Methode `CVInvoke.GetPerspectiveTransform(PointF[] src, PointF[] dest)` benutzt jeweils vier Fixpunkte, um den Warp anzuwenden. Die Punkte sind jeweils in der Reihenfolge oben links, oben rechts, unten links und unten rechts. Als Zielpunkte sind für gewöhnlich die gegebenen Bildmaße zu verwenden, also z.B.

```
var destinationPointsMat = new PointF[4];
destinationPointsMat[0] = new PointF(0, 0);
destinationPointsMat[1] = new PointF(320, 0);
destinationPointsMat[2] = new PointF(0, 240);
destinationPointsMat[3] = new PointF(320, 240);
```

Nun werden die festgelegten Anfangspunkte auf die oben genannten Endpunkte “gezerrt”. Die benötigten Werte können mit einem externen Programm (Kalibrierungsprogramm s.u.) ermittelt und direkt in der Config gespeichert werden. Eine manuelle Änderung der Werte ist über das Hauptprogramm nicht vorgesehen.
Um nun das restliche Bild zuzuschneiden -- damit nur das Spielfeld weiter in der Bilderkennung behandelt wird -- kann über EmguCV einfach ein Point of Interest (POI) gesetzt werden. Auch diese Werte werden über das Kalibrierungsprogramm ermittelt.

Um die Bilderkennung zu verbessern, wird das Bild zuerst geglättet. Per

```
CVInvoke.Medianblur(Mat src, Mat dest, int size)
```

der dritte Parameter entspricht der Kernelsize, also über wie viele Pixel der Median gebildet wird. Wir haben mit dem Wert 3 recht gute Erfahrungen gemacht. Auch andere Tutorials zur Bilderkennung benutzten diesen Wert.
Als nächstes erfolgt die Farbfilterung. Wir haben uns für die Puckfarbe grün entschieden, da sie eine noch nicht verwendete Farbe im Spielgeschehen ist. Die Filterung erfolgt in Zwei Schritten

```
CvInvoke.CvtColor(Mat src, Mat dest, ColorConversion.Bgr2Hsv);
CvInvoke.InRange(Mat src,new ScalarArray(new MCvScalar(40, 33, 30)),new
ScalarArray(new MCvScalar(85, 255, 255)), Mat dest);
```

Der erste Schritt ist ist das Konvertieren des BGR Bildes in ein HSV-Bild. Für Informationen dazu siehe
https://de.wikipedia.org/wiki/HSV-Farbraum
Der nächste Schritt ist die Filterung. Per `new MCvScalar(40, 33, 30)` werden hier Werte für den unteren und oberen Grenzwert für die Farbe grün festgesetzt. Der erste Parameter ist der Farbwinkel (hue), der zweite der Wert für die Farbsättigung (saturation) und der dritte für den Hellwert (value). Da sowohl das Umgebungslicht, als auch die Kamera das Grün der

Pucks abfälschen können, war es wichtig, die Grenzwerte nicht zu eng zu wählen. Das resultat ist ein komplett schwarzes Bild, bei dem nur die gefilterten Werte weiß dargestellt werrden. Um einen Anhaltspunkt für die einzelnen Parameter zu bekommen empfiehlt sich das Programm GIMP mit der Pipettenfunktion. Wir haben Kamerabilder abgespeichert und anschließend mit der Pipette in GIMP ausgewählt, um die HSV Werte zu erhalten. Hier sollte man acht geben, da die einzelnen Werte bei HSV von unterschiedlichen Programmen unterschiedlich interpretiert werden können. So hatte GIMP zum Beispiel einen Farbwinkel von 180°, während OpenCV mit 360° arbeitet. Hier ist also unter Umständen eine Umrechnung von Nöten.
Um nun entstandene Artefakte zu reduzieren und um die womöglich harten Kanten für die Bilderkennung zu verbessern, wird mithilfe des Befehls

```
CvInvoke.GaussianBlur(Mat src, Mat dest, new Size(9, 9), 2, 2);
```

eine Unschärfe über das Bild gelegt.
Die eigentliche Bilderkennung erfolgt in zwei Schritten. Zuerst werden mithilfe von

```
CvInvoke.Canny(Mat src, Mat edges, 50, 150);
```

Kanten im Bild erkannt. Für eine Erklärung der Parameter eignet sich die Seite https://docs.opencv.org/3.0-beta/doc/py_tutorials/py_imgproc/py_houghlines/py_houghlines.html
Der zweite Schritt ist dann die Kreiserkennung mit dem Aufruf

```
CircleF[] circles = CvInvoke.HoughCircles(edges, HoughType.Gradient, 1,
greenHueImage.Rows / 8, 100, 20, 5, 10);
```

Für eine Erklärung der Parameter eignet sich folgende Seite gut:
https://docs.opencv.org/2.4/doc/tutorials/imgproc/imgtrans/hough_circle/hough_circle.html
Zu guter letzt wird der erkannte Punkt als Koordinate gesetzt.

# Kalibrierungsprogramm

Um das Ermitteln der korrekten Parameter für das Warpen des Bildes zu vereinfachen, haben wir eine Kalibrierungssoftware geschrieben. Diese ist ebenso unter Bitbucket zu finden. Wie bereits oben erwähnt, funktioniert das Warpen sowie zuschneiden mit 4 Punkten, zwei Offsets sowie der Spielfeldhöhe und -breite.
Im Programm selber kann man die bestehende Konfigurationsdatei laden, welche dann überschrieben wird, wenn Änderungen abgespeichert werden. Als erstes muss man den Kameraindex auswählen, unter welchem die Kamera ansteuerbar ist.
Anschließend hat man die Möglichkeit, per Vorschaubild die geänderten Parameter zu testen. Der rote Rahmen repräsentiert die das Spielfeld, welches ausgeschnitten wird. Dieses kann mit den Parametern für die Spielfeldhöhe und -breite sowie den Offsets angepasst werden.
Ziel sollte es sein, dass das Spielfeld im Bild parallel zu den roten Orientierungslinien ist und das rote Rechteck das Spielfeld komplett einschließt.
Zu guter Letzt kann man die Parameter dann noch für einen Kameraindex abspeichern.

# Motorenansteuerung

Die Motoransteuerung wird für jede Achse von einem Arduino Uno mit CNC Shield (http://www.handsontec.com/dataspecs/cnc-3axis-shield.pdf) und A4988 Stepper Treibern (https://www.pololu.com/product/1182) übernommen. Am CNC Shield wird das Microstepping durch Jumper konfiguriert, wir haben 1/8 Steps genutzt. Das CNC Shield hat eine DC Eingangsspannung-Kennzeichnung von 12V-36V, wichtig ist jedoch, dass das Maximum an Betriebsspannung der A4988 schon bei 35V liegt und dieser Grenzbereich eher gemieden werden sollte. Das CNC Shield hat für jede Achse zwei Anschlüsse für End Switches, die anschließend unter den Arduino Pins 9, 10 und 11 zu finden sind, wichtig ist, dass diese Pins, wenn End-Switches angeschlossen sind, auch als Input mit Pull-Up Resistor konfiguriert werden. Die A4988 Stepper Treiber werden im Betrieb sehr heiß, was ihre Leistung beeinträchtigt. Zwar sorgt der Tisch im normalen Spielbetrieb für einen starken Luftzug, dennoch wäre zu überlegen, die Motorentreiber in Zukunft zusätzlich zu kühlen. Eine besondere Hitzeentwicklung haben die Stepper Treiber logischerweise bei Dauerlast, daher werden die Motoren zwischen den Bewegungen spannungsfrei geschaltet, was kein Problem ist, da Tisch und Motoren, auch wenn letztere die Kraft nicht halten, genug Wiederstand haben, dass sich der Schläger nicht bewegt. Das CNC Shield bietet zu den drei ansteuerbaren Achsen noch einen vierten Anschluss, an dem eine Achse gespiegelt werden kann, was für die Nutzung eines zweiten Motors auf der Längsachse des Tisches sicher hilfreich ist. Für den Arduino Code wurden einige Teile der JJulio AHRobot Bibliothek (https://github.com/JJulio/AHRobot/tree/master/Arduino) genutzt. Wichtig ist hier die direkte Nutzung der Port Register, da nur so ein ausreichend schneller Stepping Pulse erzeugt werden kann. Die Konfiguration der maximalen Beschleunigung und Geschwindigkeit ist derzeit etwas unter dem am Tisch möglichen Maximum, da ansonsten schon kleine Veränderungen am Widerstand des Schlägers zu einem „Hängen“ der Motoren führen. Mit einem insgesamt leichteren Aufbau wären rein durch die Ansteuerung nun erheblich höhere Geschwindigkeiten möglich. Die Arduinos nehmen per seriellen Kommando Positionen von der Spielberechnung entgegen. Es ist möglich, durch ein neues Kommando eine Bewegung abzubrechen, der alte Zielwert wird überschrieben und die aktuelle Bewegung abgebrochen. Da es immer wieder zu verlorenen Steps kommt, ist es wichtig, dass sich der Schläger kalibriert. Dies geschieht sowohl beim Start der Arduinos als auch bei einem entsprechenden Aufruf über die Spielberechnung, dabei wird mit geringerer Geschwindigkeit bis zu den End-Switches gefahren und somit die korrekte Null-Position des Schlägers festgestellt.

# Probleme

## Kameraerkennung

### PS3 Kameras

Die PS3-Kameras sind zwar günstig in der Anschaffung, haben aber massive Probleme mit der Treiberunterstützung. Entweder man ersetzt diese durch eine gescheite Kamera, was hohe Kosten verursacht, oder man probiert eine serielle Ansteuerung, was mit hohem Aufwand verbunden ist, welcher in der Projektplanung berücksichtigt werden muss.

### Auflösung

Ein weiteres, großes Problem war die geringe Auflösung von nur 320x240 px. Dies hat zur Folge, dass der Puck um bis zu 4 px “flatterte”, was mitunter massive Abweichungen in der Flugbahnberechnung zur Folge hatte.

## EmguCV

EmguCV ist insgesamt ein guter und mächtiger Wrapper für OpenCV. Die Codedokumentation ist aber sehr spärlich. Hier schafft die Dokumentation von OpenCV Abhilfe, die man mal mehr und mal weniger gut auf die Methodensignatur von EmguCV übertragen konnte.
Auch das Debugging ist unter EmguCV extrem schwierig, da die Fehlermeldungen sehr unpräzise sind und man oftmals eher raten musste, was der Fehler war.

## Windows und Linux

Im Hinblick auf die Oberfläche, der Kameras sowie des Bilderkennungsframeworks war es mitunter schwierig, unter Windows zu entwickeln und das Programm unter Linux zu testen. Hier hätte vielleicht von Anfang an ein zweites Linux-Betriebssystem geholfen.

## Mechanik des Tisches

Leider bremst die Mechanik des Schlägers diesen stark ab und hat mitunter Stabilitätsprobleme. Überlegungen wären hier ein Schläger aus dem 3D-Drucker (leichter, an einem Stück, eventuell mit “Federung”, da Tischplatte durch Luftzug nicht an allen Stellen gleich hoch); sowie der Ersatz aller beweglichen Holzteile durch leichtere und aller mechanischen Elemente durch leichtgängigere Alternativen.