Nutzereingaben Validation durch Objekterkennung

Kim-Eric Schiefke

Februar 2021

Zusammenfassung

Diese Projektdokumentation beschreibt eine Nutzereingabenvalidierung mithilfe von Objekterkennung. Das Ziel ist es, Computer-Maus Bewegungen in einem Kamera-Feed anhand eines Machine Learning Ansatzes zu erfassen. Die erkannte Bewegung soll vergleichbar mit den tatsächlich auf dem Computer erhaltenen Bewegungsdaten gemacht werden, um eine Manipulation der Daten auszuschließen. Für die Objekterkennung wurde ein Convolutional Neural Network (CNN) trainiert, welches auf Google MobileNetV2 basiert. Der Versuchsaufbau besteht aus einem Windows-Computer für die Inferenz und einer Webcam, welche eine Maus auf einem Mauspad filmt. Mittels manuell gelabelter Trainingsbilder war es möglich, langsame Mausbewegungen problemfrei zu erkennen. Jedoch stellt sich heraus, dass suboptimale Lichtbedingungen und die in dem Zusammenhang von der Kamera erfasste Bewegungsunschärfe den Ansatz zur Nutzereingabenvalidierung in seinem derzeitigen Zustand unbrauchbar machen.

1 Einleitung

Um die Fairness gegenüber anderen Nutzern sowohl im E-Sport als auch im Online-Casino Bereich zu gewährleisten, werden weitreichende Software-Maßnahmen getroffen. Diese Software-Maßnahmen bewegen sich heutzutage bereits mit Treibern im Kernelraum des Betriebssystems, um Betrug zu unterbinden[5]. Sie sollen validieren, dass Nutzereingaben, wie zum Beispiel Mauseingaben, nicht manipuliert oder automatisiert werden. Die Entwicklung der letzten Jahre zeigt jedoch, dass rein softwareseitige Lösungen nicht immer ausreichend sind. Mittlerweile greifen Betrüger auf FPGA-Hardware-[2] und Bilderkennungslösungen [10] zurück, welche die Möglichkeit bieten, den Schadcode auch auf einem getrennten Attacker-Computer auszuführen. Auf diesem Weg können Nutzereingaben manipuliert werden, bevor sie den Hostcomputer erreichen. Dieses Projekt versucht diesem Problem mithilfe von Objekterkennung entgegenzuwirken. So soll die Mausbewegungen getrennt von dem auf dem Hostcomputer erhaltenen Bewegungsdaten erfasst werden, um Abweichungen in den Daten zu erkennen.

2 Aufbau

2.1 Trainingsumgebung

Um MobileNetV2 auf den gewünschten Datensatz zu trainieren, wurde Google Colab genutzt. Wie aus Abbildung 1 hervorgeht, wurden die Trainingsdaten und das bereits trainierte Modell dem Colab Python Notebook über Google Drive bereitgestellt. Durch die Nutzung von Colab konnte das Transferlernen des Modells auf einer kostenfrei nutzbaren NVIDIA Tesla K80 durchgeführt werden.

In Abbildung 2 ist die verwendete Ordnerstruktur zu erkennen, wobei insbesondere der Ordner 'Workspace/embedded-ML' relevant ist. In den 'pre-trained-models' Ordner wird das Ausgangsmodell, in diesem Fall MobileNetV2, bereitgestellt. Des Weiteren wird hier die Konfigurationsdatei des Modells angepasst, in Kapitel 3.1 wird weiter auf die Konfiguration des

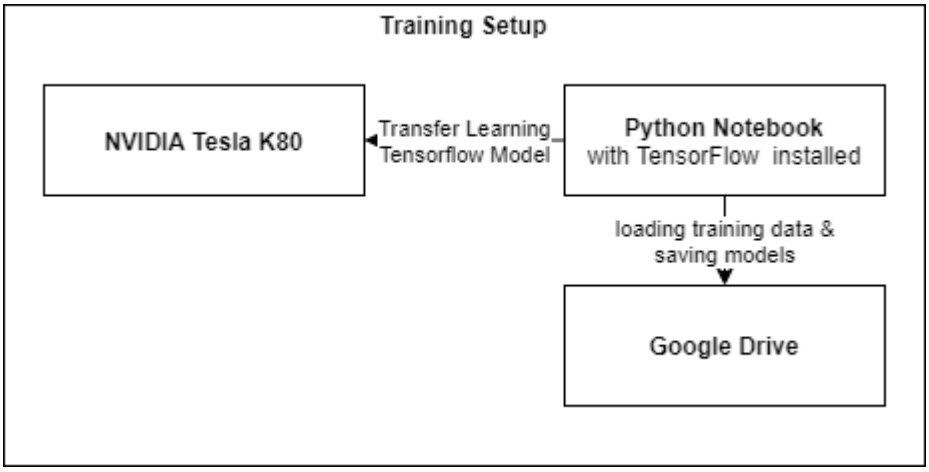

Abbildung 1: Trainingsumgebung in der Google Cloud

```
Google Drive
├── models
├── scripts
└── workspace
    └── embedded-ML
        ├── annotations
        ├── exported-models
        ├── images
        │   ├── train
        │   └── test
        ├── models
        └── pre-trained-models
```

Abbildung 2: Ordnerstruktur der Trainingsumgebung in Google Drive

Trainings Bezug genommen. Der 'models' Ordner wird verwendet, um Zwischenstände des Trainings abzuspeichern und auf verschiedene Versionen des Modells zugreifen zu können. Nach dem abgeschlossenen Training wird das transfergelernte Modell exportiert 'exported-models' Ordner exportiert. Die benötigten Trainingsdaten für die Anpassung der Objekterkennung werden in Form von handgelabelten Bildern bereitgestellt und sind in Trainings- und Testdaten unterteilt. Die dazugehörigen Label befinden sich im 'annotations' Ordner.

Abbildung 3: Vier verschiedene Bilder der Trainingsdaten

Die Trainingsdaten sind in einer Auflösung von 640*480 Pixeln mit einer Logitech C920 aufgenommen (siehe Abbildung 4 in Kapitel 2.2). Wie in Abbildung 3 sichtbar, besteht der Datensatz aus einer von vier verschiedenen Computer-Mäusen, welche jeweils in Kombinationen der folgenden Szenarien abgebildet wurden:

- Weisser Hintergrund
- Schwarzer Hintergrund
- Vollständig sichtbar
- Von einer Hand verdeckt
- Teilweise ausserhalb des Abbildungsbereich
- verschiedene Belichtungsszenarien

Bei der Erstellung der Trainingsdaten wurde darauf geachtet, einen homogenen Datensatz aus den obigen Szenarien bereitzustellen, um einem möglichen Overfitting vorzubeugen. Nach dem Erstellen der Daten wurden diese in die für MobileNetV2 benötigte Auflösung von 312*312 Pixeln transformiert.

2.2 Hardware der Testumgebung

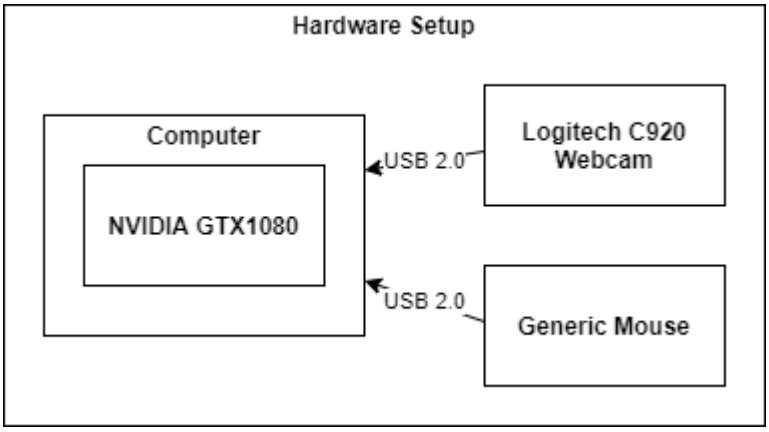

Abbildung 4: Hardware Aufbau der Testumgebung

In der Abbildung 4 ist der generelle Aufbau der Testumgebung dargestellt. Der wichtigste Bestandteil des Systems ist der Computer, welcher eine NVIDIA GTX1080 zur Beschleunigung der Inferenz nutzt. Auf dem Computer ist auch der Software-Stack zum Ausführen des Projektes installiert, dies wird weiter in dem Kapitel 2.4 beschrieben. Mit dem Computer verbunden ist eine USB-Maus, welche als Eingabegerät dient. Des Weiteren ist eine Webcam, die Logitech C920, verbunden. Die Webcam ist in einem 45° Winkel und in einer Höhe von 25 cm vor dem Aufnahmebereich aufgestellt. Mit einem 4:3 Seitenverhältnis beträgt der gemessene Aufnahmebereich der Webcam $1024\,\mathrm{cm}^3$.

2.3 MobileNetV2 als Grundlage

Wie bereits zuvor geschildert, begrenzt sich diese Dokumentation auf das Transferlernens eines CNN. Im Fall dieser Arbeit wurde sich für die MobileNetV2 Architektur mit Single Shot Detector (SSD) entschieden. Gerade die kurzen Inferenzzeiten machen MobileNetV2 lohnenswert für das Projekt.

Input	Operator	t	c	n	s
$224^2 \times 3$	conv2d	-	32	1	2
$112^2 \times 32$	bottleneck	1	16	1	1
$112^2 \times 16$	bottleneck	6	24	2	2
$56^2 \times 24$	bottleneck	6	32	3	2
$28^2 \times 32$	bottleneck	6	64	4	2
$14^2 \times 64$	bottleneck	6	96	3	1
$14^2 \times 96$	bottleneck	6	160	3	2
$7^2 \times 160$	bottleneck	6	320	1	1
$7^2 \times 320$	conv2d 1x1	-	1280	1	1
$7^2 \times 1280$	avgpool 7x7	-	-	1	-
$1 \times 1 \times 1280$	conv2d 1x1	-	k	-	

Abbildung 5: "Jede Zeile beschreibt eine Sequenz von einer oder mehreren identischen Layern, welche sich n mal wiederholen. c beschreibt die Anzahl der Ausgabekanal eines Layers, wahrend t den Erweiterungsfaktor und s das stride-Parameter beschreiben."[9]

In Abbildung 5 sind die einzelnen Layer der MobileNetV2 Architektur zu erkennen. Die Architektur startet mit einer Convolutional Layer, welche 32 verschiedene Filter beinhaltet. Auf die Convolutional Layer folgen 19 Inverted-Residual-Linear-Bottleneck Layer. Diese Form der Bottleneck Layer erlauben eine effiziente Speichernutzung, während die linearen Aktivierungsfunktionen der Layer die beste Performance im Vergleich zu z. B. einer ReLU Aktivierungsfunktion innerhalb der MobileNetV2 Architektur bietet[9].

2.4 Softwareumgebung und Projekt

Die Softwareumgebung basiert auf einer Anaconda Environment mit einer Installation von Google Tensorflow. Neben Tensorflow wurde openCV verwendet, um Zugriff auf die Webcam zu erhalten. Der Zugriff erfolgte, wie bereits bei den Trainingsdaten in einer Auflösung von 640*480. Um einem IO-Bottleneck entgegenzuwirken, wurde das Aufnehmen der Bilddaten über die

Klasse WebcamFeedGrabber in einen eigenen Thread ausgelagert. Zum Aufnehmen der Bewegungsdaten der Maus wurde die Softwarebibliothek mouse genutzt[4]. Diese Bibliothek bietet einen Python Wrapper zur Nutzung der Microsoft Low Level Mouse Hooking API[7].

```
def mouseCallback(mouseEvent):
    if not hasattr(mouseEvent, 'x') or not hasattr(mouseEvent, 'y'):
        return
    global lastPosition
    global movement

    movement['x'] += mouseEvent.x - lastPosition['x']
    movement['y'] += mouseEvent.y - lastPosition['y']
    lastPosition['x'] = mouseEvent.x
    lastPosition['y'] = mouseEvent.y
```

Abbildung 6: Funktion zum erfassen der Mausdaten

Zum Verarbeiten der Bewegungsdaten des erstellten Maushooks wurde ein Callback (siehe Abbildung 6) erstellt. Der Callback berechnet anhand der im Mausevent übertragenen Positionsdaten das Positionsdelta zur vorherigen Position, dies geschieht asynchron zu unserem Main-Loop des Projektes. Nach der Initialisierung unseres Modells für die Inferenz wird Main-Loop gestartet. Jeder Durchlauf beginnt mit der Entnahme der aktuellsten Webcam-Aufnahme aus dem WebcamFeedGrabber. Zu diesem Zeitpunkt wird das Ergebnis des Positionsdeltas abgespeichert und die movement Variable aus Abbildung 6 auf x, y = 0 zurückgesetzt. Die entnommene Aufnahme wird in die für das Modell erforderliche Auflösung skaliert, diese beträgt 312*312. Nach der Skalierung wird die Inferenz durchgeführt. Wenn eine Maus auf der Aufnahme erkannt wurde wird, wie bereits zuvor bei dem aufgenommenen Positionsdelta, der Mausbewegung wird hier das Positionsdelta der erkannten Maus berechnet und abgespeichert. Mit dem Stoppen des Main-Loops stehen somit 2 Arrays derselben Länge zur Analyse Verfügung, das eine mit der erfassten Mausbewegung und ein weiteres mit auf den Kamera-Aufnahmen erkannten Mausbewegung.

3 Hauptteil

3.1 Training des Modells

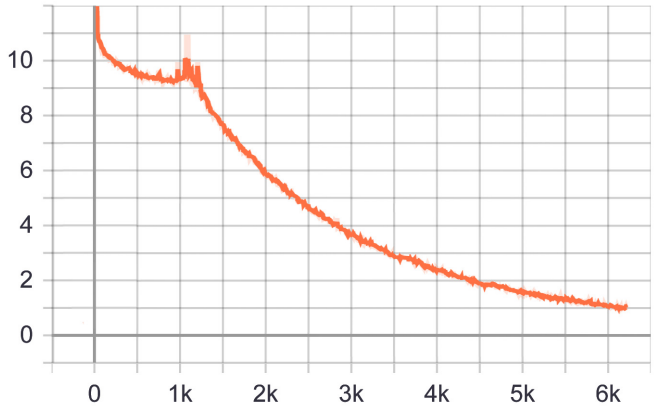

Abbildung 7: Trainingsverlust der totalen Verlustfunktion des Transferlernens von MobileNetV2 mit dem neuen Datensatz

Das Ausgangsmodell wurde mit einer Batch-Größe von 16 für 100'000 Schritte trainiert. Das bedeutet, dass das Modell mit 6250 Batches mit jeweils 16 zufällig ausgewählten Bildern aus dem Trainingsdatensatz transfergelernt wurde. Die Größe des Datensatzes beträgt 300 Trainingsbilder, das bedeutet, dass das Modell im Durchschnitt ungefähr 21 Mal mit dem gesamten Datensatz trainiert wurde. In der Abbildung 7 ist der Trainingsverlauf zu erkennen.

3.2 Testen des Modells in der Testumgebung

Im Projektrahmen wurde das transfertrainierte Modell in Hinblick auf zwei Kernpunkte die Performance untersucht. Die Testumgebung wurde in Kapitel 2.2 beschrieben. Zum einen wurde getestet, ob die Inferenz des Modells auf einer gebräuchlichen Grafikkarte ausreichend schnell ist. Des Weiteren wurde die Genauigkeit des Modells bei der Objekterkennung untersucht.

Inferenzzeiten in ms auf GTX1080			
Anzahl	Durchschnitt	Minimum	Maximum
100	27.38	22.33	31.26
1000	24.47	21.04	32.30
2000	23.12	20.31	32.54

Tabelle 1: Inferenzzeiten in der Testumgebung, jede Zeile zeigt den Mittelwert aus 3 Durchläufen

In Hinblick auf die zeitliche Performance des Modells wurden Inferenzen anhand eines in der Testumgebung erstellten Videos durchgeführt. Mit einer Länge von einer 1m10s stehen bei 30 Bildern pro Sekunde 2100 verschiedene Bilder für die Inferenz zur Verfügung. In der Tabelle 3 sind die Erbenisse für drei verschiedene Anzahlen von Inferenzen dargestellt. Alle Inferenzen wurden in derselben festen Reihenfolge des Videos durchgeführt, das bedeutet, dass ein einzelner Testlauf keine Bilder mehrfach verarbeiten kann. Aus den Messungen geht hervor, dass das Modell unter optimalen Bedingungen in unserer Testumgebung 43 Bilder in einer Sekunde verarbeiten kann. Diese Messungen ignorieren den weiteren Rechenaufwand, welcher durch IO-Zugriffe und Bildtransformierungen entsteht.

Um die Genauigkeit des Modells zu Untersuchen wurden fünf verschiedene Szenarien untersucht. Es wurde ein Durchlauf jedes Szenarios mit jeweils 1000 Inferenzen durchgeführt, um jeweils eine Tabelle der zu erstellen.

Testen der Genauigkeit des Modells			
Szenario	Inferenzen	korrekt	Genauigkeit
Maus ohne Hand (statisch)	1000	967	96.7 %
Maus ohne Hand (bewegt)	1000	464	46.4 %
Maus mit Hand (statisch)	1000	887	88.7 %
Maus mit Hand (bewegt)	1000	406	40.6 %
Hand ohne Maus (statisch)	1000	830	83 %

Tabelle 2: Verschiedene Testszenarien um die Genauigkeit des hier vorgestellten Modells aufzuzeigen

In der Tabelle 2 ist zu erkennen, dass die Genauigkeit des Modells deutlich abnimmt, sobald sich das zu erkennende Objekt in Bewegung findet. Diese Daten deuten auf mehrere Probleme in diesem Projekt hin. Bei der Erstellung

des Trainingsdatensatzes wurden nur statische Objekte gelabelt, was die Erkennung bewegten Objekten deutlich erschwert. Das zweite Problem ist die Kombination aus Lichtbedingungen und den Einstellungen der verwendeten Webcam. Durch nicht optimale Lichtbedingungen werden von der Logitech C920 fehlende Lichtinformationen durch das Erhöhen der Verschlusszeiten kompensiert. Das bedeutet, dass die erhaltenen Aufnahmen ein großes Maß an Bewegungsunschärfe aufweisen und sich nicht zum labeln eigenen. Des Weiteren zeigt sich, dass wenn keine Maus vorhanden ist die Genauigkeit des Modells sinkt und auch eine Hand als Maus erkennt.

3.3 Analyse der Projektdaten

Die Datenanalyse wurde in drei Schritten durchgeführt. Im ersten Schritt wurden jeweils 1000 Datenpunkte der Objekterkennungs-Inferenzen und 1000 Datenpunkte des Maushooks gespeichert. Gleichzeitig wurden die aufgenommene Bildreihe abgespeichert, um Aufschluss über die Korrektheit der Daten zu liefern. Im zweiten Schritt wurden die Daten visualisiert, um zu schauen, ob sich aus der Visualisierung weitere Informationen ergeben. Letzteres wurden die Datenpunkte manuell auf Auffälligkeiten verglichen.

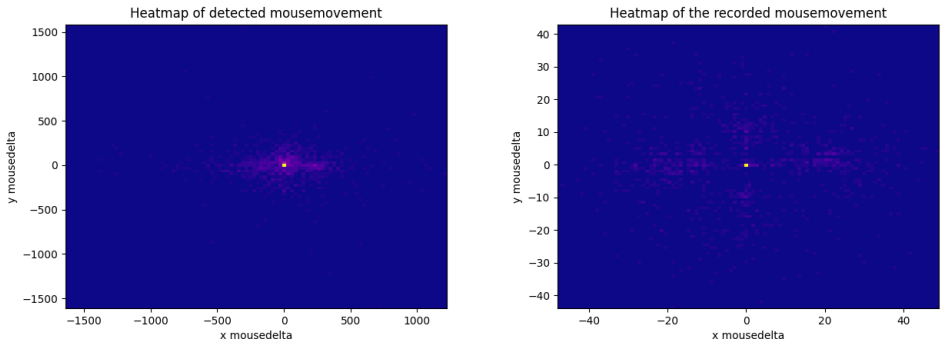

Abbildung 8: Plots von 1000 Datenpunkten des selben Durchlaufs. Links die erkannte Mausbewegungs, rechts die tatsächliche Mausbewegung

In der Abbildung 8 stellt sich heraus, dass sich durch ein Plotten der aufgenommenen Datenpunkte sich in beiden Datenquellen dasselbe Muster abzeichnet. Es zeigt, dass durch die Ungenauigkeiten bei den Inferenzen bereits

einzelne Werte erfasst werden, welche sich deutlich außerhalb des erwarteten Wertebereichs befinden. Einige Datenpunkte des Bilderkennungsplots erreichen auf der x-Achse Werte von unter -1400, obwohl sich der erwartete Datenbereich zwischen -500 und 500 befindet. Dies stellt eine deutliche Einschränkung für das Projekt dar, da dies darauf ausgelegt ist, zuverlässig Auffälligkeiten in den Mausbewegungen zu erkennen.

Aufgenommene Mausbewegungen				
	Maushook		Bilderkennung	
Inferenz Nr.	X-Achse	Y-Achse	X-Achse	Y-Achse
1	0	0	24.14	-0.61
2	0	0	21.04	2.05
3	0	0	-0.74	2.057
4	0	0	-7.43	3.81
990	8	9	58.99	78.45
991	7	9	85.67	94.82
992	8	10	111.17	136.77
993	12	6	80.22	101.22
994	24	0	58.66	15.45
995	31	-2	339.74	-34.82
996	25	-1	159.01	16.77
997	18	-1	423.12	5.49
998	16	-1	187.06	-12.78
999	14	-3	252.79	11.22

Tabelle 3: Auschnitt aus einer Tabelle der aufgenommenen Mausbewegungen im zeitlichen Verlauf gegenübergestellt.

Beim Auswerten der Tabelle 3 zeigt sich, dass es Zeiträume gibt (Inferenz Nr. 1-4), in welchen der Maushook keine Daten erfasst. Dies ist damit zu erklären, dass die Maus von dem Nutzer angehoben wird. Das bedeutet, dass die sogenannte Lift-Off-Distance(LOD) der Maus, welche die maximale Aufzeichnungsdistanz des Maussensors beschreibt, überschritten wurde. Des Weiteren zeigt sich, dass Ungenauigkeiten bei der Objekterkennung das hier angewandte Verfahren für kleinere Bewegungen unbrauchbar machen. Es lässt sich eine Tendenz der jeweiligen Richtung der Maus erkennen, doch es bestehen deutliche Abweichungen (Inferenz Nr. 990-999). Wie bereits zuvor

bereits in Kapitel 3.2 erläutert, sind die Daten auch bei schnellen Mausbewegungen ab ca. 12 cm/s unbrauchbar, da das derzeitige Modell Bewegungsunschärfe in einem Bild nicht verarbeiten kann.

4 Ergebnisse

4.1 Ansätze zur Problemlösung

4.1.1 Überschrittene Lift-Off-Distance

Es gibt zwei grundlegende Ansätze mit der überschrittenen LOD umzugehen. Der naive Ansatz ist es, alle Daten der Objekterkennung zu ignorieren, wenn sich im selben Zeitraum keine Bewegungsänderungen durch die Daten des Maushooks ergeben. Dieser Ansatz widerspricht dem Grundkonzept des Projektes, da aus den Daten nicht hervorgeht, ob die LOD wirklich überschritten wurde oder ob eine Manipulation der Daten vorliegt, bevor Sie den Hostcomputer erreichten.

Der zweite Ansatz ist zu erkennen, ob die LOD überschritten wurde. Für die Umsetzung gibt es verschiedene Möglichkeiten. Dem derzeitigen Modell kann eine weitere Klasse Namens 'computer_mouse_liftoff' hinzugefügt werden. Da die generelle LOD von Enthusiasten-Mausperipherie im Bereich von 0.5-2 mm liegt, ist dieser Ansatz gerade in Verbindung mit Bewegungsunschärfe und ohne aufwendige Kalibrierung und Rechenoperationen zur Kompensation der Kameraperspektive nicht realistisch[3]. Eine weitere Problemlösung kann der Einsatz einer zweiten Kamera sein. Die Kamera muss auf der Höhe der Maus parallel zur horizontalen Ebene aufgestellt sein. Die Daten dieser Kamera können mittels eines transfergelernten Klassifizierungsmodells genutzt werden, um zwischen den Klassen 'no_lift_off' und 'lift_off' zu unterscheiden.

Im Vergleich zu dem naiven Ansatz wird deutlich, dass für die Erkennung der LOD weitere Ressourcen in Form von Hardware und Rechenleistung beansprucht werden, welchen die Umsetzung für die weitreichende Verwendung fraglich gestaltet.

4.1.2 Bewegungsunschärfe und Genauigkeit

Um den Ungenauigkeiten des Modells vorzubeugen, gibt es verschiedene Ansätze. Zum einen besteht die Möglichkeit, den Trainingsdatensatz zu erweitern. Durch einen größeren Datensatz kann eine generelle Verbesserung der Bilderkennungsperformance des Modells in verschiedenen Umweltbedingungen erreicht werden. Durch eine Erweiterung des Datensatzes kann jedoch nicht die Bewegungsunschärfe der Kamera kompensiert werden, weil eine Unterscheidung von verschiedenen Gegenständen bei hoher Bewegungsunschärfe nicht möglich ist.

Ein weiterer Ansatz ist, Transformationen auf dem Eingangsbild zu verwenden, um der Bewegungsunschärfe entgegenzuwirken[6]. Dieser Weg bedeutet jedoch einen höheren Rechenaufwand und beeinträchtigt die zeitliche Performance dieses Projekts.

Um der Bewegungsunschärfe entgegenzuwirken, ist die Voraussetzung eine niedrigere Verschlusszeit der Kamera. Eine Möglichkeit stellt eine Ersetzung der hier verwendeten Logitech C920, durch eine Kamera mit einem Bildsensor, welcher größere Fotozellen besitzt und mit einer lichtstarken Linse ausgestattet ist. Ein weitere Möglichkeit ist die Verbesserung der Lichtbedingungen. Unter optimalen Lichtbedingungen ist es auch mit der hier vorhandenen Logitech C920 möglich, die Verschlusszeiten zu verringern und die Bewegungsunschärfe zu minimieren. Jedoch kann die Verwendung von lichtstarken Lampen in der Testumgebung störend auf den Nutzer einwirken.

Keiner der angesprochenen Lösungswege ist kosteneffizient und ohne Einschränkungen für den Nutzer umzusetzen.

4.2 Ein alternativer Ansatz ohne Objekterkennung

Bei der Suche nach einem Ansatz, welcher eine bessere Alternative zu dem hier vorgestellten Verfahren darstellt, scheint dies die Tracking-Technologie, welche auch bei Virtual Reality Hardware verwendet wird, zu sein. Bei diesem Tracking-Verfahren gibt es mindestens zwei Basisstationen und einen aktiven Tracker, welcher über Infrarot-Strahlung mit den Basisstationen in-

teragiert. Aus einer Untersuchung der Technischen Universität von Lissabon in Zusammenarbeit mit einem NASA-Forschungszentrum geht hervor, dass sich bei einem Abstand von 1-2 Metern von den Basisstationen zum Tracker nur wenige mm bis cm Abweichungen zur idealen Erfassung des Objektes ergeben[1]. Durch das Vermeiden der Objekterkennung und die Möglichkeit zur exakten Erfassung der Position des Trackers würde die Möglichkeit bestehen, mithilfe von Machine-Learning Modellen die erfassten Bewegungsdaten auf Anomalien im Vergleich zu den in diesem Projekt erfassten Maus-Hook-Daten zu untersuchen. Diese Form des Trackings funktioniert mit einer Latenz von im schlechtesten Fall nur wenigen ms, um der Simulatorkrankheit vorzubeugen. Das HTC Lighthouse Tracking stellt eine Verbesserung der zeitlichen Performance im Vergleich zu dem hier vorgestellten Verfahren bis zum Faktor 10 dar[8].

5 Fazit

Generell lässt sich das hier vorgestellte Projekt als nicht erfolgreich einstufen. Es ist gelungen, eine generelle Objekterkennung einer Maus durch Transferlernen zu bewerkstelligen. Im Verlauf des Projektes haben sich jedoch die gravierende Nachteile der hier vorgestellten Methode gezeigt. So haben sich die technischen Limitierungen der hier verwendeten Webcam und nicht optimalen Lichtverhältnisse bemerkbar gemacht. In Kapitel 4.2 wird ein Alternativansatz beschrieben, welcher das hier vorgestellte Verfahren selbst unter optimalen Bedingungen deutlich übertrifft. Im Vergleich zur Alternative stehen die Kosten und der Aufwand zum Erreichen der optimalen Bedingungen in diesem Projekt in keinem Verhältnis.

Literatur

[1] *HTC Vive: Analysis and Accuracy Improvement*, Madrid, Spain, 2018. IEEE.

[2] *BMCLeech: Introducing Stealthy Memory Forensics to BMC*, volume 32 of *Forensic Science International: Digital Investigation*, Amsterdam, Niederlande, 2020. Elsevier.

[3] SteelSeries ApS, 2018. `https://steelseries.com/gaming-mice/rival-600#specs`.

[4] boppreh. mouse: Hook and simulate global mouse events in pure python, 2017. `https://github.com/boppreh/mouse`.

[5] ESL Gaming GmbH, 2014. `https://play.eslgaming.com/germany/anti-cheat`.

[6] T. N. Jha. Velocity detection from a motion blur image using radon transformation, 2010.

[7] Microsoft. Lowlevelmouseproc callback function, 2018. `https://docs.microsoft.com/en-us/previous-versions/windows/desktop/legacy/ms644986(v=vs.85)`.

[8] okreylos. Lighthouse tracking examined, 2016. `http://doc-ok.org/?p=1478`.

[9] M. Sandler. Mobilenetv2: Inverted residuals and linear bottlenecks, 2019.

[10] Trombov. Futurennaimbot: Universal neural network aimbot for all games with custom training mode, 2019. `https://github.com/Trombov/FutureNNAimbot`.