Praktikum Copmuter Vision

Smart Camera Operator

Vorgelegt von:

Tobias Johanning, Markus Konetzny, Tabea Preusser

Münster, den 05.03.2020

Inhaltsverzeichnis

1. Einleitung

1.1. Motivation

Im Rahmen eines Projektes des Unternehmens Rimondo sollte ermöglicht werden, dass die Abdeckung der Videoaufnahme von Reitsportturniere maximiert werden soll. Dazu sollen neben den internationalen auch von möglichst vielen, im besten Falle sogar allen, nationalen Turnieren in NRW Videoaufnahmen erstellt werden können, was später auch auf ganz Deutschland erweitert werden soll. Neben Bereitstellung der Turnieraufnahmen in der Mediathek von Rimondo sind auch Live-Aufnahmen in der Zukunft geplant. Alleine 2018 fanden in NRW über 600 nationale und 82 internationale Reitsportturniere statt, von denen nur ein Bruchteil verfilmt wurde. Dabei stellt das grundlegende Problem die hohen Personalkosten der Kameraleute dar, die für stundenlange professionelle Aufnahmen benötigt werden. Während dies für internationale Turniere, wie das „Turnier der Sieger“ in Münster, möglich über Fernsehsender zu finanzieren ist, sind die Kosten für die kleineren, regionalen Turniere nicht rentabel. Da jedoch die Kundengruppe, welche aus den Turnierteilnehmern und deren Fans besteht, mit über 98.000 Mitgliedern in Reitsportvereinen in NRW als mögliche Kunden, großes Potenzial darstellt, wird eine Lösung benötigt. Aus diesem Grund wurden Projektgruppen von Informatikstudenten der WWU vor die Aufgabe gestellt die Kameraführung für Reitsportturniere zu automatisiert, indem ein „Smart Camera Operator“ im Rahmen eines Praktikums erstellt wird. Dies sollte eine Software umfassen, welche in der Lage ist den Kameramann für Turnieraufnahmen im Pferdesport zu ersetzen.

1.2. Aufgabenstellung und Zielsetzung

Die Erstellung des „Smart Camera Operators“ zur Videoaufnahme von Reitsportturnieren sollte in drei Phasen unterteilt erreicht werden, wobei die Wahl der Methoden und Verfahren freigestellt wurde.
Die erste Phase hatte die Erstellung von Trainingsdaten von Pferden und Reitern zum Ziel, womit die Vorarbeit für die weitern Phasen geleistet wurde. Dazu hat Rimonod auf einer Online Platform ca. 1300 Frames aus Reitvideos zur Verfügung gestellt, auf denen mithilfe eines Labeling-Tools die Position der Reiter und Pferde eingetragen werden sollte. Jeder Teilnehmer des Praktikums hatte vom 24.10.2019 bis zum 14.11.2019 Zeit in den Bildern mindestens 200 Reiterpaare, genauer 400 Reiter oder Pferde, zu kennzeichnen. Diese Daten

wurden anschließend von Rimondo ausgewertet und in Form einer Datenbank für die zweite Phase des Projektes weiterverwendet.
Die zweite Phase, vom 19.11.2019 bis 19.12.2019, umfasste das Training eines Detektors anhand der zuvor erstellten Datenbank. Dieser soll in der Lage sein anhand von Reitern und Pferden auch Reiterpaare zu erkennen. Als Grundlage zum Training des Detektors wurde mehrstündiges Videomaterial von Weitwinkel-Kameras von Reitern aus derselben Reithalle zum Testen zur Verfügung gestellt. Es sollen erste Videos erstellt werden, die anhand der Region of Interest (Roi) aus den Frames der detektierten Reiterpaare erstellt werden, wobei noch kein großer Wert auch flüssige Bildübergänge gelegt werden musste. Zudem soll dieser Detektor genutzt werden, um weitere Bilddaten zu labeln und den Detektor mit diesen Daten weiter zu trainieren.
Unser persönliches Ziel in dieser Phase ist einen bereits robusten Tracker mit dem bereitgestellten Videomaterial zu ermöglichen, weshalb wir direkt mit Maschine Learning beginnen wollen. Folglich lag der Fokus mehr auf der Detektion von Reiter und Pferd als konkret auf dem Reiterpferdepaar. Wir wollten direkt bei der Bestimmung der Rois besonders die Randfallbehandlung des Sichtfeldes einbeziehen und zunächst alle Reiterpaare gemeinsam im Bildausschnitt haben. Weiter wollten wir den Aspekt des Labelling weiterer Daten, für Videos und Bilder mithilfe einer GUI umsetzen.
In der dritten Phase soll der endgültige „Smart Camera Operator“, vom 20.12.2019 bis 6.3.2020 erstellt werden. Es wird dazu weites Videomaterial zur Verfügung gestellt, welches mehrere Reithallen Indoor und Outdoor abdeckt. Dabei soll die genauere Lokalisierung des Reiterpaars für eine robustere Implementierung des Trackers genutzt werden. Außerdem soll das Tracking der Rois flüssiger werden, wobei beispielsweise ein Kalmanfilter sowie Entzerrung des Bildes eingesetzt werden können. Unser Ziel war es ein Flüssiges Video mithilfe eines passenden Filters zu erreichen, wobei wir einen Gaußfilter ausprobieren wollten. Weiter wollten wir den Reiterpaar Detektor verbessern, indem wir Sprünge und Verschwinden eines Paares einbeziehen, wodurch Probleme wie das Auftauchen im Spiegel gelöst werden sollen. Ebenfalls sollte das Verdeckungsproblem des beobachteten Paares in Angriff genommen werden, um ein einzelnes Reiterpaar erfolgreich zu tracken. Weiter sollte der Detektor durch weitere Daten aus Indoor und Outdoor Aufnahmen erweitert werden, um Vielseitigkeit zu erlangen

1.3. Aufbau der Arbeit

In Kapitel 2 gehen wir auf die grundlegenden Entwurfsentscheidungen bezüglich verwendeter Hard- und Software sowie den Computer Vision Methoden ein. Weiter werden in Kapitel 3 der Anwendungsaufbau mit den genutzten Verfahren und die Benutzeroberfläche genauer beleuchtet. Die Implementierung des Detektors und Trackers wird in Kapitel 4 erläutert und anschließend werden

die erreichten Ziele in Kapitel 5 diskutiert. Zum Schluss fasst Kapitel 6 das erfolgte Projekt zusammen und gibt einen Ausblick auf mögliche Verbesserungen dafür wichtige Schwerpunkte.

2. Entwurfsentscheidungen

2.1. Programmiersprache

Für die Wahl der Programmiersprache haben wir zunächst die Vorkenntnisse aller Gruppenmitglieder in verschiedenen erlernten Sprachen abgeschätzt und betrachtet welche Sprachen einen einfachen Einstieg in das Thema Maschine Learning erlauben. Deshalb viel unsere Wahl trotz geringerer Geschwindigkeit im Vergleich zu C++ auf Python, da ohne viel Vorwissen schnell ein erster funktionierender Prototyp erstellt werden kann und wir zudem unserer Kenntnisse in dieser Sprache vertiefen können.
Für Python gibt es eine Vielzahl geeigneter Entwicklungsumgebungen wie Eclipse oder Visual Studio, die Wahl fiel jedoch auf die kostenlose Variante von PyCharm, mit der alle Gruppenmitglieder vertraut waren.

2.2. Setup

Mit dem Ansatz im Verlauf des Projektes Maschine Learning zu verwenden, wurde recht schnell deutlich wie wichtig eine gute Hardware Ausrüstung ist, um gute Performance beim Training des Models und bei der Detektion zu erreichen. Während die Umsetzung auch ausschließlich mit CPU möglich ist, besticht der Einsatz von GPU mit deutlicher Geschwindigkeit. Um diesen Vorteil zu nutzen, haben wir uns entschlossen für das Training mit dem kostenlosen Cloud-Service von Google Colaboratory in Verbindung mit Google Drive zu arbeiten, der ebenfalls kostenlos GPU Nutzung ermöglicht. Dabei stehen uns 25 GB Ram und je nach Zuweisung eine Tesla T4 GPU mit ca.8 GB oder eine Tesla K80 GPU mit ca. 12 GB zur Verfügung, was einen 25-fachen Geschwindigkeitsvorteil von GPU gegenüber CPU darstellt. Passend zur Wahl der Programmiersprache arbeitet Google Colab mit Jupyter Notebooks und hat bereits die meisten Bibliotheken installiert, wobei fehlende mit Kommandozeilen Befehlen noch hinzugefügt werden können.

2.3. Versionsverwaltung

Um Änderungen an Dateien und Quellcode zu erfassen und sinnvoll zu strukturieren, bietet sich aufgrund von Zusammenarbeit mehrerer Gruppenmitglieder der Einsatz einer Versionsverwaltung an. Durch Organisation mit Zeitstempeln

und Benutzerkennungen kann gemeinsam an Dateien gearbeitet, Änderungen nachvollzogen, Dateien wiederhergestellt und Zugriffe koordiniert werden. Die Wahl, mit welcher Versionsverwaltung das Projekt umgesetzt werden sollte, fiel auf Git als verteiltes System, welches wir in Form von Github nutzen. Dies hat den Grund, dass Git von der genutzten Entwicklungsumgebung PyCharm unterstützt wird und alle Mitglieder unserer Gruppe bereits Github durch vorherige Projekte vertraut waren, sodass relativ wenig Einarbeitungszeit erforderlich war. Die Einbindung in Google Colab ist mit Github ebenfalls möglich, indem das jeweilige Projekt geclont wird, was wir für die Datenbank und Maschine Learning benötigten.

2.4. Toolselection

Opencv Numpy Scipy Scikit-learn tensorflow Pandas keras

2.5. Maschine Learning

hier oder im nächsten Kapitel?

3. Anwendungsaufbau

3.1. Klassendiagramm

3.2. Designpattern

MVC und Observer in GUI

3.3. GUI

QT5 Design css stylesheet

3.4. Klassische Computer Vision

3.5. Maschine Learning

4. Implementierung

4.1. Phase 1

4.2. Phase 2

4.2.1. Training des Model

Bevor wir mit dem Training des Detektors angefangen haben, mussten wir uns anhand einiger Tutorials die Verwendung von Mask Rcnn anlesen und an einigen Beispielen testen.
Im ersten Schritt haben wir die bereitgestellten Daten aus der Datenbank von Phase 1 aufbereitet, um diese zum Training zu verwenden. Dazu haben wir die Datenbank aufgeteilt, sodass pro Frame eine csv Datei mit allen Labeln im Format "image,label,x,y,width,height"existierte. Da jeder Frame mehrfach gelabelt wurde, haben wir mithilfe von Schwellwerten die zusammengehörigen Dopplungen bestimmt und davon den durchschnittlichen Wert abgespeichert. Anschließend haben wir die hohe Auflösung der Frames verringert, um die Trainingszeit zu verringern. Zuletzt haben wir die Daten in einer passenden Ordnerstruktur von "accepted_images"und dem darin liegenden Ordner "annotations"in ein Github Projekt eingebunden, sodass das Training von Google Colab aus erfolgen kann.
Im zweiten Schritt haben wir die benötigten Klassen zum Training erstellt. Dazu wurde die Klasse RiderConfig als Unterklasse der Config Klasse von Mask Rcnn erstellt, in der die Parameter individuell angepasst wurden. Wichtig war die Anzahl der zu detektierenden Klassen, die neben Reiter und Pferd auch den Hintergrund umfasst. Zudem wurde die Leistung der verfügbaren GPU angepasst sowie die Trainings- und Validierungsschritte pro Epoche. Weiter wurde die Klasse RiderDataset als Unterklasse der Dataset Klasse von Mask Rcnn erstellt. In dieser wurden die nötigen Funktionen zum Laden des Datensatzes in Trainings und Test Modus, zum Laden der Masken und extrahieren der Boxen anhand von Eckpunkten aus den cvs Dateien überschrieben.
Damit konnten wir im dritten Schritt das Model trainieren, wozu wir aufbauend auf den MS COCO Gewichten mittels Transfer Lernen Zeit sparen konnten. Für das Training haben wir den Datensatz in 70% Trainings-,15 % Test- und 15% Validierungsdaten unterteilt. Die erste Version unseres Detektors haben wir mit 75 Epochen mit 500 Schritten trainiert, bis der Verlust pro Epoche sehr gering wurde. genauer

4.2.2. Extraktion der Rois

4.2.3. weiters Labeln

4.3. Phase 3

4.3.1. GUI

4.3.2. weiteres Training des Model

4.3.3. Reiterpaar

4.3.4. Sprünge/Verschwinden

4.3.5. Verdecken

5. Ergebnis

5.1. Planung

5.2. Umsetzung GUI

5.3. Umsetzung Detektor

5.4. Umsetzung Tracker

6. Schlussfolgerung und Ausblick

A. Videos Ergebnisse

B. Videos Siegerehrung