



Praktikum Computer Vision

bei Jun.-Prof. Dr. Benjamin Risse

## Smart Camera Operator

Vorgelegt von Gruppe 6 :

Tobias Johanning  
421503  
t\_joha02@wwu.de  
M.Sc. Informatik

Markus Konetzny  
406462  
m\_kone06@wwu.de  
M.Sc. Informatik

Tabea Preusser  
428026  
t\_preu04@wwu.de  
M.Sc. Informatik

Münster, den 05.03.2020



# Inhaltsverzeichnis

<b>1. Einleitung</b>	<b>1</b>
<b>2. Entwurfsentscheidungen</b>	<b>3</b>
2.1. Setup . . . . .	3
2.2. Toolselection . . . . .	4
2.3. Machine Learning . . . . .	5
<b>3. Anwendungsaufbau</b>	<b>6</b>
3.1. Programmaufbau . . . . .	6
3.2. Video-Pipeline . . . . .	7
3.3. Zustandsautomat . . . . .	8
<b>4. Implementierung</b>	<b>9</b>
4.1. Phase 1 . . . . .	9
4.2. Phase 2 . . . . .	9
4.3. Phase 3 . . . . .	11
<b>5. Ergebnisauswertung</b>	<b>15</b>
5.1. Planung . . . . .	15
5.2. Umsetzung und Zielerreichung . . . . .	15
5.3. Ausblick . . . . .	17
<b>6. Zusammenfassung</b>	<b>18</b>
<b>Anhang A. Videos</b>	<b>19</b>
A.1. Ergebnisse . . . . .	19
A.2. Siegerehrung . . . . .	19

# 1. Einleitung

## Motivation

Im Rahmen eines Projektes des Unternehmens Rimondo sollte die Abdeckung der Videoaufnahme von Reitsportturniere maximiert werden. 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 Zukunft geplant. Alleine 2018 fanden in NRW über 600 nationale und 82 internationale Reitsportturniere<sup>1</sup> 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, für Fernsehsender finanzierbar ist, ist dies 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.

## 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 weiteren Phasen geleistet wurde. Dazu hat Rimondo 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

---

<sup>1</sup>Daten der Deutschen Reiterlichen Vereinigung E.V.

## 1. Einleitung

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 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 auf flüssiges Tracking gelegt werden musste. Zudem soll dieser Detektor genutzt werden, um weitere Bilddaten schnell zu labeln und ihn mit diesen Daten weiter zu trainieren.

In der dritten Phase, vom 20.12.2019 bis 6.3.2020 soll der endgültige *Smart Camera Operator*, erstellt werden. Es wurde dazu weiteres Videomaterial zur Verfügung gestellt, welches verschiedene Reithallen und einen Reitplatz abdeckte. Dabei soll die genauere Lokalisierung des Reiterpaars für eine robustere Implementierung des Trackers genutzt werden. Außerdem soll das Tracking der ROI flüssiger werden, wobei beispielsweise Filter sowie Entzerrung des Bildes eingesetzt werden können.

Unser persönliches Gruppenziel ist es einen robusten und vielseitigen Tracker mit Machine Learning zu erstellen, der in der Lage ist einem einzelnen Reiterpaar zu folgen. Dabei liegt unser Fokus auf Problemen wie Verdeckung und Kreuzen von Reitern in unterschiedlichen Umgebungen, während wir uns weniger mit dem Perfomance Aspekt befassen wollen. Zudem wollen wir im Hinblick auf Benutzerfreundlichkeit das Labeling weiterer Daten und das Konvertieren der Videos mithilfe einer GUI umsetzen.

## Aufbau der Arbeit

In Kapitel 2 gehen wir auf die grundlegenden Entwurfsentscheidungen des Projektes ein, wobei die verwendete Hard- und Software sowie der Machine Learning Ansatz vorgestellt werden. Weiter wird in Kapitel 3 der Anwendungsaufbau anhand der Funktionsweise des Detektors sowie der Benutzeroberfläche beleuchtet. Die Implementierung der einzelnen Phasen des Trackers wird anschließend in Kapitel 4 im Detail erläutert. Die erreichten Ziele und möglichen Verbesserungen mit dafür wichtigen Schwerpunkten werden in Kapitel 5 diskutiert und zum Schluss fasst Kapitel 6 das erfolgte Projekt zusammen.

# **2. Entwurfsentscheidungen**

## **2.1. Setup**

### **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 Machine Learning erlauben. Deshalb fiel 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, wir haben uns jedoch für die kostenlose Variante von PyCharm entschieden, mit der alle Gruppenmitglieder vertraut waren.

### **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. 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.

### **Hardware**

Mit dem Ansatz im Verlauf des Projektes Machine 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

## 2. Entwurfsentscheidungen

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. Die Einbindung von Github Projekten in Google Colab ist ebenfalls möglich, was wir für die Nutzung der Datenbank und Machine Learning benötigten.

### 2.2. Toolselection

Für die Implementierung in Python wurden fachspezifische Bibliotheken eingesetzt, von denen wir die wichtigsten hier aufführen wollen

**Pandas** Zum lesen, sortieren und aufteilen der Datenbank haben wir die Bibliothek Pandas verwendet, die effizient in der Lage ist diese Daten zu manipulieren, zu filtern und mit fehlenden Werten umzugehen.

**OpenCV** Das bekannteste Computer Vision Framework OpenCV besticht mit seiner Vielzahl an Algorithmen, deren Schnelligkeit durch das C++ Backend und Benutzerfreundlichkeit durch den Python Wrapper besticht. Zum Einsatz kommt OpenCV sowohl bei der Extraktion von Frames, beim Filtern als auch durch die Zeichenfunktionen.

**Numpy** Für wissenschaftliche Berechnungen kam Numpy bei großen Bilddatensätzen, bei Vektorrechnungen und dem Versuch mit Ausgleichskurven zu arbeiten zum Einsatz.

**Scipy** Die SciPy Library bietet neben Numpy eine breite Masse an verschiedenen mathematischen Werkzeugen und numerischen Algorithmen. Besonders die vordefinierte Funktion für den eindimensionalen Gausfilter half beim Smoothing aller Bounding Boxen.

**Tensorflow** Das Framework TensorFlow bietet ein umfangreiches Grundgerüst für Machine-learning mit dem es möglich ist, auf Machine-learning basierende Programme einfach zu erstellen und zu verwenden.

**Keras** Ursprünglich war Keras teil der Tensorflow Core API, jedoch wird es als eigenständige Bibliothek weitergeführt. Durch eine einheitliche Schnittstelle sorgt Keras für einen erheblich einfacheren und benutzerfreundlichen Umgang mit Tensorflow. Die Kombination aus Keras und Tensorflow sorgte für einen einfachen und schnellen Einstieg in die Welt der auf Machine Learning basierenden Programme.

## 2. Entwurfsentscheidungen

**PyQt** Für die Gestaltung der grafischen Nutzeroberfläche haben wir uns entschieden PyQt5 einzusetzen, da uns diese Bibliothek bereits aus der Programmiersprache C++ in ähnlicher Form bekannt war. Zudem besticht diese im Vergleich zu Alternativen wie TKinter durch den Ansatz die GUI vom Backend zu trennen und die Möglichkeit ein modernes Design zu erstellen.

## 2.3. Machine Learning

Da die Verwendung von Python im Bereich Machine Learning populär ist, gibt es viele Frameworks, welche die Klassifikation oder Detektion von Objekten in Bildern unterstützen.

Wir haben uns aus folgenden Gründen für Mask R-CNN<sup>1</sup> entschieden:

- **Zuverlässige Detektion:**

Auch unter schwierigen Bedingungen verspricht M-RCNN eine präzise, zuverlässige Detektion. Auch wenn die Geschwindigkeit nicht an Alternativen wie YOLO heranreicht, ist die Qualität der Klassifikation und Detektion immer sehr hoch.

- **Leichter Einstieg:**

Auch wenn es subjektiv (noch) nicht so weit verbreitet ist wie YOLO fanden sich zwar wenige - dafür aber sehr gute Quellen zum Einstieg inklusive Tutorials und jupyter notebooks im Git repository der Entwickler.

- **Segmentierung:**

M-RCNN kann Segmentierungsmasken berechnen, mit denen wir die Qualität unseres Models gut beurteilen konnten. Auch wenn wir die Masken letztendlich noch nicht verwendet haben, erschien es uns als sehr vorteilhaft, diesen Trumpf in der Hinterhand zu haben.

- **Interesse:**

Beim Stöbern haben wir gesehen, dass M-RCNN ein relativ neues Verfahren ist. In dem Wissen, dass die meisten anderen Gruppen sich für YOLO entscheiden würden, sind wir nicht zuletzt auch deshalb den Weg mit M-RCNN gegangen, um zu herauszufinden, wie es sich im Vergleich zu YOLO verhält. Sinn und Ziel des Praktikums ist es ja auch gewesen, Neues auszuprobieren, zu experimentieren und aktuelle Verfahren sowie state-of-the-art-Techniken kennenzulernen.

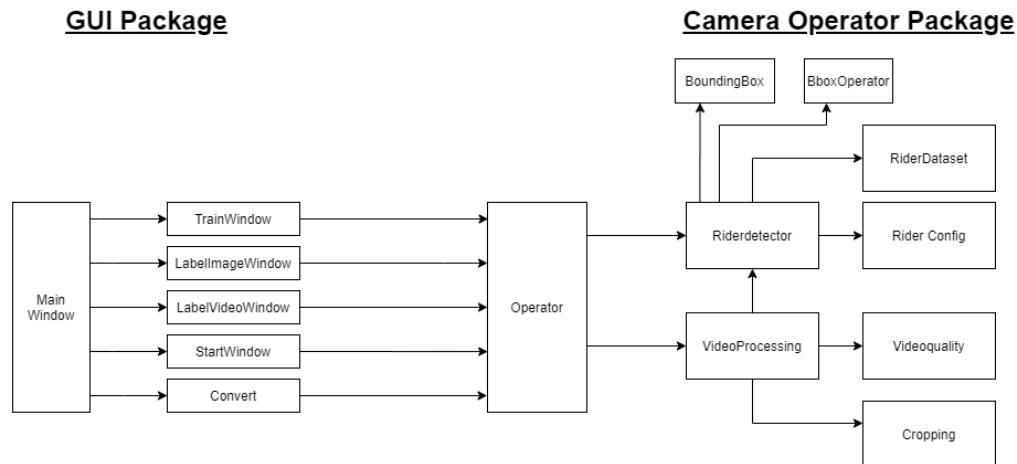
---

<sup>1</sup>Mask R-CNN Github Projekt

# 3. Anwendungsaufbau

Im Folgenden wollen wir den Aufbau der implementierten Software bezüglich ihrer Funktionalität und der getroffenen Designentscheidungen der autonomen Kameraführung betrachten.

## 3.1. Programmaufbau

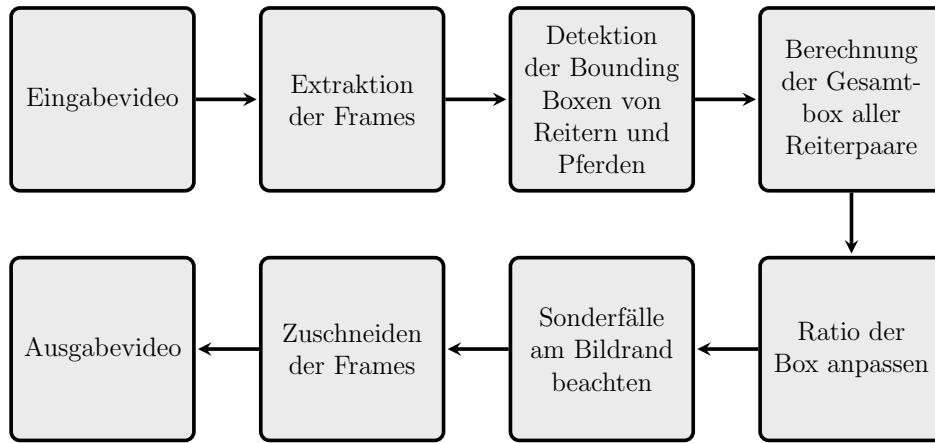


**Abb. 1.:** Klassendiagramm vom Smart Camera Operator

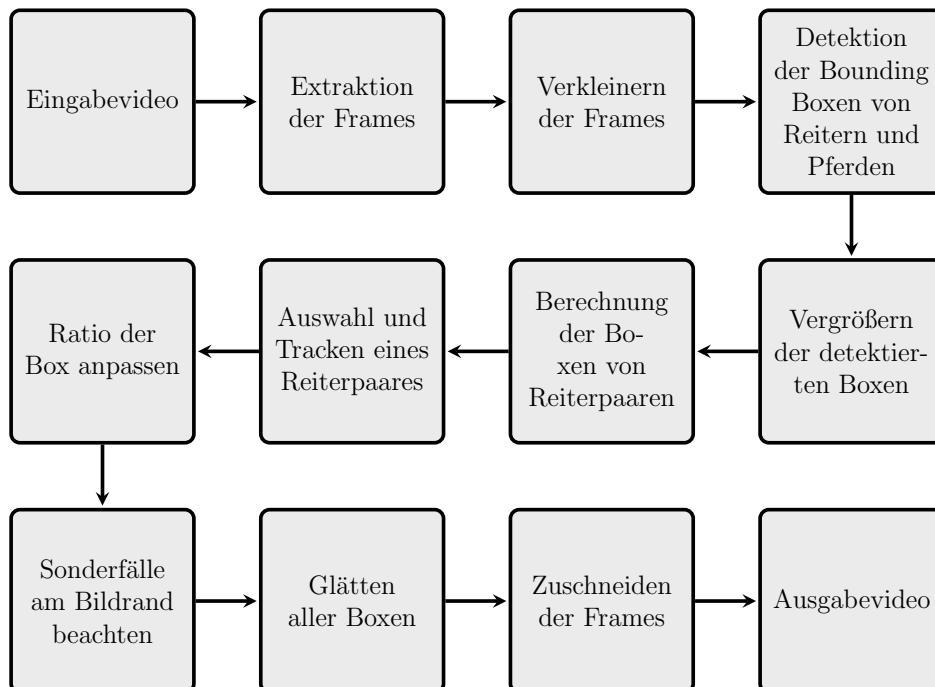
Wir haben Frontend und Backend separiert, wobei die Klasse des Operators die Schnittstelle bildet. Für die Umsetzung der GUI wurde das Prinzip des Model View Controllers verwendet, um die Nutzereingaben effizient verarbeiten zu können. Der Smart Camera Operator besteht aus den beiden Hauptkomponenten des Detektors, der Training und Object Detection übernimmt, sowie des Videoverarbeitung, welche das Tracking eines Reiterpaars und die Videoqualität umfasst.

### 3. Anwendungsaufbau

## 3.2. Video-Pipeline



**Abb. 2.:** Video-Pipeline Phase 2



**Abb. 3.:** Video-Pipeline Phase 3

### 3. Anwendungsaufbau

Anhand der Ziele von den Phasen zwei und drei haben wir die Vorgehensweise mit einer Video-Pipeline geplant, welche wir dann schrittweise mit sinnvoller Aufgabenteilung umsetzen konnten. Die erste Version (Abb. 2) umfasst dabei lediglich die grundlegenden Funktionen, die für das sinnvolle Bestimmen der ROI nötig sind. So wird hier pro Frame eines Videos für alle detektierten Reiterpaare ein gemeinsamer Bildausschnitt berechnet und diese ROI zu einem Ausgabevideo zusammengefügt. Diese Video-Pipeline konnten wir dann in der dritten Phase (Abb. 3) um weiter Aspekte erweitern, wobei diesmal der Fokus auf der Qualität des Ausgabevideos lag. Die Detektion konnte auf Frames mit geringerer Auflösung schneller durchgeführt werden, weshalb die Größe der gefundenen Boxen anschließen angepasst werden musste. Anstatt dem vorherigen Ansatz alle Reiterpaare zu verfolgen, wird nun ein einzelnes davon ausgewählt. Die Qualität des Ausgabevideos wird durch weniger sprunghafte Unterschiede in der Größe der aufeinanderfolgen ROI verbessert, sodass der Bildfluss flüssiger wird.

### 3.3. Zustandsautomat

Folgender Moore-Automat (Abb. 4) soll zeigen, wie unser Smart Camera Operator idealerweise funktionieren soll. Der Automat startet im *missing* Zustand. Die Transitionen *gefunden* und *nicht gefunden* beziehen sich auf ein Reiterpaar, das im Bild gefunden wurde - oder nicht. In der unteren Hälfte jedes Zustands befinden sich die Aktionen, die durchgeführt werden solange sich der Automat in dem Zustand befindet.

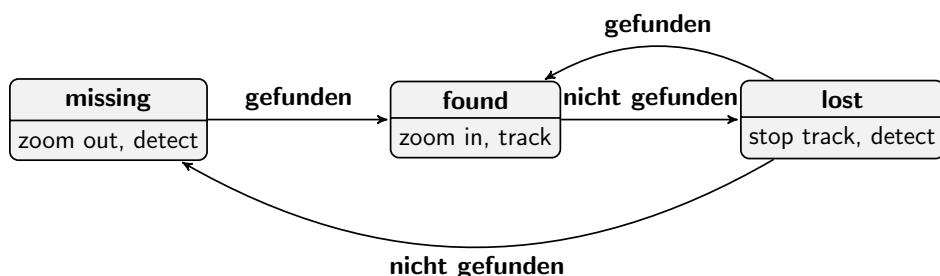


Abb. 4.: Zustandsautomat vom Smart Camera Operator

# 4. Implementierung

## 4.1. Phase 1

Wie vorgesehen hat jedes Gruppenmitglied die Online-Platform von Rimondo genutzt um auf Bildern einer Reithalle 400 Reiter oder Pferde zu markieren. Dazu wurden Bounding Boxen um die erkannten Reiter bzw. Pferde eingezeichnet und ein passendes Label zur Unterscheidung hinzugefügt. Die gesamten erstellten Daten wurden am Ende der Phase von Rimondo in eine Datenbank umgewandelt mit welcher in der nächsten Phase der geplante Detektor entwickelt werden sollte.

## 4.2. Phase 2

### Training des Detektors

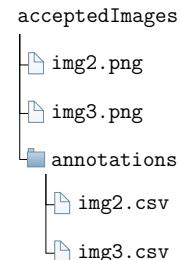
Bevor wir mit dem Training angefangen haben, mussten wir uns in die Thematik von Image Detection sowie in die Verwendung von Mask R-CNN einlesen.

Im ersten Schritt haben wir die bereitgestellten Daten aus der Datenbank aufbereitet, um diese zum Training zu verwenden. Dazu wurden die Daten so aufgeteilt, sodass pro Frame eine csv Datei mit allen Labeln und relativen Koordinaten existierte. Da jeder Frame mehrfach gelabelt wurde, konnten wir mit einem Threshold die zusammengehörigen Dopplungen bestimmen und davon den durchschnittlichen Wert abgespeichern.

Anschließend wurde die hohe Auflösung der Frames verringert, um die Trainingszeit zu verkürzen.

Zuletzt haben wir die Daten in einer passenden Ordnerstruktur (Abb.5) von *acceptedImages* und dem darin liegenden Ordner *annotations* in ein Github Projekt eingebunden, sodass das Training von Google Colab aus erfolgen konnte.

Im zweiten Schritt wurden die benötigten Klassen zum Training mit Mask R-CNN erstellt. Dazu ist die Klasse RiderConfig als Unterklasse der Config Klasse erstellt worden, in der die Trainingsparameter 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



**Abb. 5.:** Ordnerstruktur  
Training

#### 4. Implementierung



**Abb. 6.:** Segmentierung von Reiter und Pferd nach dem Training der zweiten Phase

verfügbar GPU sowie die Trainings- und Validierungsschritte pro Epoche angepasst. Weiter wurde die Klasse RiderDataset als Unterklasse der Dataset Klasse erstellt. Die nötigen Funktionen zum Laden eines Datensatzes im Trainings- und Testmodus und zum Laden der Masken, in Form der extrahierten Boxen anhand von Eckpunkten aus den cvs Dateien, wurden darin überschrieben.

Damit konnten wir im dritten Schritt mit dem Training des Models beginnen, wozu wir ausgehend von den COCO<sup>1</sup> Gewichteten Transfer Lernen verwendet haben. Für das Training haben wir unseren Datensatz in 70% Trainings-, 15 % Test- und 15% Validierungsdaten unterteilt. Die erste Version unseres Detektors haben wir mit 75 Epochen mit 500 Schritten trainiert. Da Mask R-CNN den Loss nicht nur aus den Bounding Boxen, sondern auch aus den Masken berechnet, deren Genauigkeit für die Implementierung zu vernachlässigen waren, war ein Verlust unter 10 % pro Epoche unser Zielwert. Die damit erreichte Leistung unseres Detektors war für die bereitgestellten Videos ausreichend zuverlässig (Abb. 6), sodass wir danach mit diesem die ersten Videos erstellen konnten.

Zur vielfältigen Nutzung des erstellten Detektors, haben wir das Training mit weiteren gelabelten Bilddaten ermöglicht, um den Detektor auch in anderen Hallen zuverlässiger verwenden zu können. Dazu werden die Bilder eines Ordners nacheinander angezeigt, wobei die detektierten Pferde und Reiter mit farbigen Bounding Boxen eingezeichnet werden. Anhand dieser Boxen kann der Nutzer entscheiden, ob die Daten geeignet sind, sodass die akzeptierten Bilder und deren Boxen wie in der Ordnerstruktur von Abb.5 wiederum abgespeichert werden, um den Detektor damit weiter zu trainieren.

### Berechnung der ROI

Mithilfe von OpenCV werden alle Frames des Eingabevideos einzeln durchlaufen und der zuvor trainierte Detektor auf jedes einzeln angewandt. Dadurch haben wir pro Frame alle ROI mit Bounding Boxen, als Umrandung von erkannten Reitern und Pferden, ermittelt.

---

<sup>1</sup>Common Object in Context

#### 4. Implementierung

Diese Menge aus ROI wird nun gefiltert, da wir nur noch Reiterpaare weiter betrachten wollen. Ein Reiterpaar ist definiert als ein Mensch, der sich sehr nah an oder auf einem Pferd befindet. Da die Bounding Boxen Rechtecke sind, können wir also berechnen, wie groß der Flächeninhalt der überlappenden Rechtecke von Reiter und Pferd ist. Ein Flächeninhalt gleich Null bedeutet dann, dass sie sich überhaupt nicht überlappen. Dadurch ist es nicht nur möglich zu berechnen, ob ein Pferd nah an einem Menschen ist - sondern auch wie nah, indem wir einen Threshold gewählt haben, der die Größe des errechneten Flächeninhalts prüft. Beispielsweise hätte ein Mensch *auf* einem Pferd einen größeren Flächeninhalt als ein Mensch *neben* einem Pferd, was je nach Wunsch also angepasst werden kann. Wenn eine Mensch-ROI also eine Pferd-ROI überlappt, zeichnen wir um beide eine Reiterpaar-ROI. Damit haben wir einzelne Pferde, Reiter und Fehldetections vom Tracking ausgeschlossen.

Wir verfolgten hier den Ansatz zunächst alle Reiterpaare in den ROI abzubilden und uns erst in der dritten Phase selektiv auf ein einzelnes Paar zu fokussieren. Dieser Modus kann weiterhin gut für Aufwärmphasen der Reitturniere genutzt werden, bei denen mehrere Reiterpaare auf dem Turnierplatz sind. Die Umsetzung ist als austauschbarer Filter konzipiert, der eine Gesamtbox anhand der maximalen und minimalen Eckpunkte aller detektierten Bounding Boxen bestimmt. Ausgehend von der Gesamtbox müssen für die Erstellung erster Videos noch nötige Sonderfälle beachtet werden. So wird der Zoom unter 240 Pixel verhindert, um eine akzeptable Bildqualität bei weit entfernten Objekten zu ermöglichen. Die längere Seite der Bounding Box ist ausschlaggebend für die Berechnung der passenden Ratio des Bildausschnittes, sodass dann die optimalen Eckpunkte berechnet werden können. Wichtig ist es noch die Ränder des Sichtfeldes zu berücksichtigen und die Ratio entsprechend zu korrigieren. Im letzten Schritt werden die ROI mithilfe der kalkulierten Gesamtbox aus jedem Frame ausgeschnitten und auf dieselbe Größe skaliert, was durch die gleiche Ratio aller Frames keine Verzerrung zur Folge hat, und ein Ausgabevideo geschrieben.

### 4.3. Phase 3

#### Weiterentwicklung des Detektors

Mit Beginn der dritten Phase wurden weitere Reitvideos zur Verfügung gestellt, welche in verschiedenen Hallen sowie im Außenbereich gefilmt wurden. Der erste Test mit dem Detektor verlief nicht sonderlich zufriedenstellend, wodurch wir mehrere Probleme identifizieren konnten. Zum einen wurden nicht alle Reiter und Pferde, die weit entfernt von der Kamera waren, korrekt detektiert. Dieses Ergebnis war in dem zuvor recht einseitigen Training begründet, da sich die Hintergründe, Reiter und Pferde kaum unterschieden. Zum anderen wurden auch viele Zuschauer als Reiter identifiziert und einige Gegenstände

#### 4. Implementierung

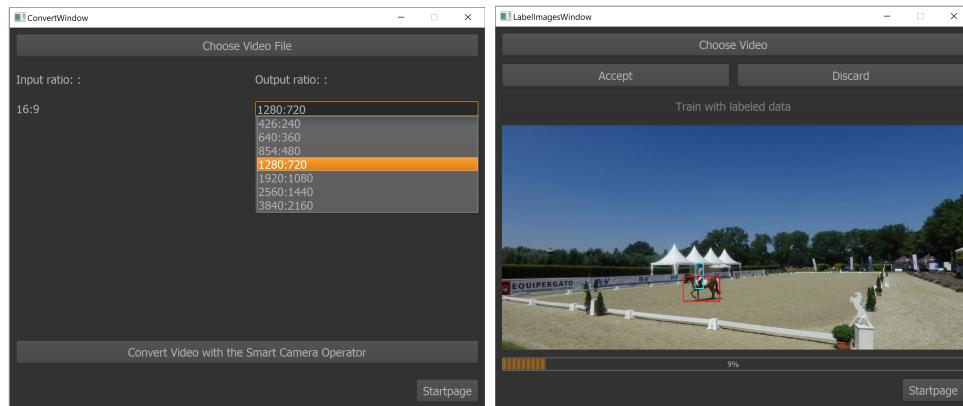


**Abb. 7.:** Segmentierung von Reiter und Pferd nach dem Training der zweiten Phase

z.B. Pflanzen fehlerhaft erkannt. Nachdem wir den Detektor jedoch mit ca. 500 weitere Frames pro neuer Umgebung weiter trainiert hatten, wurden ein Großteil der Fehler beseitigt (Abb. 7). Durch die Erweiterung des Detektors ist dieser nun auch in der Lage unterschiedlich gekleidete Reiter und Pferde mit abweichende Fellfarben zuverlässiger zu detektieren.

## Benutzeroberfläche

Im Hinblick auf die Kundengruppe haben wir die Kennzeichnung weiterer Bilddaten und die Konvertierung eines Videos für autonome Kameraführung in eine GUI eingebunden. Die GUI wurde mit PyQt5 erstellt und das Standardaussehen durch Einbinden eines Stylesheets modifiziert. Mit dieser minimalen Benutzeroberfläche kann der Nutzer nun zunächst auswählen, ob weitere Daten gelabelt oder ein Video umgewandelt werden soll (Abb.8). Bei der Auswahl weiterer Bilddaten können sowohl Ordner mit Bildern als auch Videos mithilfe extrahierter Frames verwendet werden. Dazu kann der Nutzer pro Bild entscheiden, ob die eingezeichnete Detektion akzeptiert oder abgelehnt wird und den Detektor im Anschluss damit weiter trainieren. Für die Umwandlung eines Videos wird nach Auswahl der Datei und der Auflösung des Ausgabeformates ein Video mithilfe des Smart Camera Operators erstellt.



**Abb. 8.:** Benutzeroberfläche Konvertieren und Labeln eines Videos

## 4. Implementierung

### Performance

Da der Fokus unserer Gruppe auf der Genauigkeit des Detektors liegt, haben wir nur einige grundlegende Verbesserungen der Performance vorgenommen, welche dem nicht entgegenwirken. Im ersten Schritt wird dazu anhand der Auflösung des Eingabevideos entschieden, ob die Frames vor der Detektion verkleinert werden sollten. Dies war bei den aufgenommenen Videos mit 4K Auflösung sinnvolle, da Mask R-CNN auch mit einfacher HD Auflösungen ähnlich gute Ergebnisse liefert. Somit konnte der Zeitaufwand der Detektion von ca. 0.81 auf ca. 0.48 Sekunden pro Detektion verbessert werden. Anschließend müssen die detektierten Boxen wieder entsprechend hochskaliert werden, um mit den Ursprungsbildern übereinzustimmen.

Zur weiteren Verbesserung haben wir den Einsatz von Ausgleichsgeraden und Ausgleichskurven getestet. Als Ansatz haben wir den Gedankengang verfolgt, nicht für jeden Frame Detektionen durchzuführen, sondern nur für jeden n-ten Frame, wodurch die Anwendung beschleunigt würde. In beiden Fällen haben wir mithilfe von Numpy die Berechnungen mit *polyfit* anhand der letzten 10 Detektionen erstellt. Die Ergebnisse für die Ausgleichsgerade stellten sich zwar performancemäßig als sehr positiv dar, jedoch nahm die Videoqualität stark ab. Das Ausgabevideo wies aufgrund der Ausgleichsgeraden sehr regelmäßige gerade Kameratranslationen auf, die die Gesamtqualität verminderten. Als nächstes nutzten wir statt der Geraden Ausgleichskurven, wobei sich bereits Kurven vom Grad 2 oder 3 ausreichend gut eigneten. Während dadurch die vorherigen Makel im Ausgabevideo gelöst wurden, stellte sich nur sehr geringfügiger bis gar kein Performacevorteil heraus. Aufgrund dessen wurden beide Versuche die Anzahl der Detektionen zu verringern verworfen und aus Zeitgründen nicht weiter verfolgt.

### Videoqualität

Der Schwerpunkt der Phase richtet sich auf ein robusteres und flüssigeres Tracking, was wir mit besonderen Fokus auf Ausnahmesituationen und die Qualität der Kameraführung umgesetzt haben.

### Tracking

Da der Detektor nicht komplett fehlerlos ist und demnach ein uninteressantes und entferntes Objekt für einen Zeitraum auch mal priorisieren kann, ist es wichtig diese Sprünge zu unterbinden. Ein Reiterpaar bewegt sich im Normalfall nicht innerhalb zweier Frames über den ganzen Platz. Der Abstand des Reiterpaars zu seiner Position auf dem vorherigen Bild ist also relativ gering. Um diese Sprünge zu vermeiden prüfen wir also, ob die Position des getrackten Objektes des aktuellen Bildes stark von der Position des vorherigen getrackten Objektes abweicht. Damit der Wert der vorherigen Position als relativ akkurat

#### 4. Implementierung

gelten kann, nehmen wir, statt nur die Position des vorletzen Bildes, den Median der letzten 25 getrackten Boxen. Sollte nun die Distanz einen Schwellwert überschreiten, wird die neue Box ignoriert und für das aktuelle Bild die zuletzt valide Box verwendet. Damit der Smart Camera Operator nicht auf einer Position hängen bleibt, ist ein Sicherungsmechanismus eingebaut, der ihn wieder zurücksetzt und neues Tracking erlaubt.

Sollte das Reiterpaar sich hinter einem Hindernis verstecken oder aus dem Bild hinauslaufen, wird die Kamera an der zuletzt gesichteten Stelle verbleiben bis das Paar wieder sichtbar geworden ist oder ein neues Paar detektiert wird.

#### Glättung

Um die Folge der ROI zu glätten, damit sich nach dem Zuschneiden ein sanftes Seherlebnis für den Zuschauer bietet, haben wir verschiedene Verfahren ausprobiert. Das effektivste, effizienteste und gleichzeitig das einfachste Verfahren, das zum Einsatz kam, war ein eindimensionaler Gaußfilter. Wir haben eine Folge aus ROI, dabei hat jedes ROI 4 Attribute - die Koordinaten  $(x_1, y_1), (x_2, y_2)$  der linken unteren und rechten oberen Ecke des Rechtecks. Wir verwenden für alle Koordinaten  $x_1, y_1, x_2, y_2$  aller ROI einen eindimensionalen Gaußfilter mit  $\sigma = 5$ , genauer nutzen wir also insgesamt vier Mal einen Filter über die ROI für jede der vier Koordinaten.

Eine weitere Möglichkeit wäre es gewesen, die Richtungsänderungen zu erkennen oder zu berechnen und dann einfach die ROI linear von einer Richtungsänderung zur nächsten wandern zu lassen, was sich beispielsweise mit *optical flow* berechnen lässt. Das wäre auch für die Echtzeit-Anwendung nötig gewesen, bei der die Kamerabewegung und der Position des getrackten Objektes orientieren soll. Da wir aber für unseren Anwendungsfall eine effizientere Möglichkeit gefunden haben und Live-Übertragungen den Umfang des Praktikums überschreiten, haben wir dies nicht weiter thematisiert und implementiert.

# **5. Ergebnisauswertung**

## **5.1. Planung**

Die vorgegebenen Phasen aus der Aufgabenstellung dieses Projektes stellten sich als sehr hilfreich für ein sinnvolles schrittweises Vorgehen und dessen Umsetzung heraus, besonders im Hinblick auf fehlendes Vorwissen aller Gruppenmitglieder in der Thematik Machine Learning. Während die erste Phase problemlos ablief, gestaltete sich der praktische Einstieg in der zweiten Phase dagegen relativ langwierig, bis wir einen ersten Prototypen unseres Detektors erstellen konnten. Als Folge dessen war es uns nicht vollständig möglich das Ziel eines Reiterpaardetektors bereits komplett fertigzustellen, da dieser an einigen Stellen noch Fehler aufwies, welche jedoch direkt zu Beginn der dritten Phase behoben wurden. Ebenfalls sollte in der zweiten Phase das Labeling weiterer Daten umgesetzt werden, wobei die entsprechend vereinfachte Benutzung dieses Ziels erst mithilfe der GUI in Phase drei hinzugefügt wurde. Diese zeitlichen Verschiebungen konnten jedoch mit effizienter Aufgabenteilung gut bewältigt werden und stellten kein Hindernis für die letzte Phase dar. Durch die recht allgemeine Formulierung der Ziele in Phase drei, die einen Fokus auf robustes und flüssiges Tracking legten, haben wir die daraus abgeleiteten Gruppenziele in den Vordergrund gestellt. Obwohl einige Konzepte, die aus Effizienz- oder Performancegründen verworfen wurden, erprobt wurden, konnten die Projektziele größtenteils eingehalten werden.

## **5.2. Umsetzung und Zielerreichung**

Im Folgenden betrachten wir die Umsetzung der einzelnen Ziele und die Einstufung von bestehenden Fehler sowie deren Wichtigkeit im Anwendungskontext.

### **Detection**

Der erste Schritt für die autonome Kameraführung war die Erstellung eines Detektors für Reiter und Pferde sowie später auch für Reiterpaare. Diesen konnten wir erfolgreich mithilfe von Machine Learning umsetzen, indem wir die erstellten Trainingsdaten der ersten Phase sowie Transfer Learning als Grundlage genutzt haben und den Detektor später um weitere Datensätze erweitert haben. Mit Wahl von Mask R-CNN haben wir schon am Ende der zweiten

## 5. Ergebnisauswertung

Phase ausreichende Genauigkeit erreicht, dass wir problemlos zusätzliche Daten mit weiterem Videomaterial labeln konnten, obwohl sich die Umgebung und die Reiterpaare unterschieden (siehe Video6). Die Schnelligkeit des Labelings konnten wir durch eine graphische Benutzeroberfläche gewährleisten, mit welcher ein Nutzer die möglichen Detektionen beurteilen kann. Dies stellt für den Einsatz der Software mit ungeschulten Nutzern einen großen Vorteil dar, die mit wenigen Interaktionen ein Video nach der Aufnahme konvertieren können. Trotz der erfolgten Erweiterung des Detektors ist es in jeder neuen Umgebung auch weiterhin nötig diesen weiter zu trainieren, da durchaus fehlerhafte Erkennungen auftreten können. Das Detektieren von Reiterpaaren konnten wir simpel mithilfe der Überdeckung von Reiter und Pferd lösen. Als Folge dessen konnten wir einzelne Reiter und Pferde vom Tracking ausschließen, was in neuen Umgebungen robusteres Detektieren zur Folge hat (siehe Video1).

Während unser Ziel eines robusten Detektors mit Mask R-CNN gut umsetzbar war, stellt die Performance trotz Nutzung von GPU einen Nachteil gegenüber anderen Frameworks wie YOLO dar. Im Hinblick auf Liveübertragungen müsste dieser Interessenskonflikt genauer beurteilt werden und Möglichkeit gefunden werden, dass nicht für jeden Frame Detektionen benötigt werden. Die getesteten Verbesserungen mithilfe von polynominaler Regression ergab zwar nicht die gesuchte Geschwindigkeit, das Verfahren konnte jedoch eine gute Videoqualität erzielen, weshalb sich eine weitere Überprüfung sicherlich lohnen würde.

## Tracking

Die flüssige und robuste Verfolgung des einzelnen Reiterpaars stellte sich als größere Herausforderung heraus als wir zu Anfang angenommen hatten. Um dies dennoch zu ermöglichen haben wir viele grundlegende Problematiken betrachtet und diese versucht schrittweise zu lösen. Neben der Behandlung der Bildänder haben wir auch das Verdecken und das fehlerhafte oder fehlende Detektieren von einzelnen Paaren behandelt. Nachdem wir Sprünge und Verschwinden der errechneten ROI bestimmt und einen Gaußfilter angewendet haben, konnten wir die Videoqualität zufriedenstellend verbessern. An dieser Stelle würde es sich anbieten andere Filter auszutesten um herauszufinden welcher das beste Verhältnis zwischen Rechenaufwand und Qualität leistet, der verwendete Gaußfilter hat für die Nachbearbeitung von Videos jedoch bereits eine gute Qualität geliefert.

Der erstellte Tracker weist an einigen Stellen noch geringe Fehler beim Verfolgen eines einzelnen Reiters auf, wenn die Verdeckung zu lange anhält. Wir konnten zwar sicherstellen, dass der Reiter zuverlässig nach der Verdeckung wiedergefunden wird, jedoch wird nicht immer das gewünschten Ergebnis geliefert (siehe Video2). Dementsprechend ist das Kreuzen von Reitern eine Ausprägung dieses Problems, was den für den Tracker jedoch insgesamt eine etwas geringere Herausforderung war, da die Detektion für diesen Fall bessere Er-

## *5. Ergebnisauswertung*

gebnisse lieferte (siehe Video3 und Video4). Wir mussten feststellen, dass einige Aspekte von manueller Kameraführung im Allgemeinen schwer umsetzbar sind, da beispielsweise situationsbedingt entschieden wird, welches Reiterpaar gefilmt werden soll (siehe Video2). Im Hinblick auf den Einsatz bei Reitsportturnieren sind die Einzelprüfungen jedoch der wichtigste Bereich, welchen wir am meisten optimiert haben.

### **5.3. Ausblick**

Durch die zeitlichen Vorgaben konnten nicht alle Ideen und Ansätze ausprobiert werden, jedoch sehen wir in einigen Aspekten großes Potenzial zur Verbesserung und Erweiterung.

Der erstellte Detektor für Reiterpaare sollte sowohl für weitere Umgebungen, besonders im Außenbereich mit anderen Reitern und Pferden, weiter trainiert werden, als auch für verschiedene Reitdisziplinen wie beispielsweise Springreiten oder Voltigieren, bei denen sich die Bewegungsmuster verändern. Um eine noch robustere Verfolgung eines Reiterpaars zu ermöglichen sind weitere Sonderfälle, wie das Kreuzen und längeres Verdecken von Reitern, genauer zu beachten. Ebenfalls sind Ausnahmesituationen wie Stürze von Interesse, da in diesem Fall Reiter und Pferd nicht mehr als Paar erkannt würden. Ein ausgiebiger Praxistest würde uns an dieser Stelle Aufschluss über nötige Erweiterungen und Problematiken geben.

Während für das Praktikum eine feste Position und Blickrichtung der Kamera vorgegeben waren, müsste der bestehende Smart Camera Operator im Hinblick auf den Einsatz für Live Tracking an einigen Stellen modifiziert werden, da dies andere Herausforderungen als die nachträgliche Bearbeitung von Aufzeichnungen mit sich bringt. Besonders müsste die Performance dafür deutlich erhöht werden, damit die Erkennung für ca. 25 fps erreicht werden kann. Davor ausgehend könnten dann auch Kamerabewegungen ermöglicht werden, die dem fokussierten Reiterpaar mittels Translationen und Rotationen folgt.

Die Verbesserung der Kameraführung hat noch großen Spielraum offen, bis ein professioneller Kameramann ersetzt werden kann, da diese durch ihre Erfahrung im Vorteil sind. Für einen Ausgleich könnte herausgefunden werden, an welchen Stellen sich ein Zoom auf den Reiter oder das Pferd für die jeweilige Turnierart lohnt, da Springreiten andere Anforderungen als Dressurreiten hat.

## **6. Zusammenfassung**

Im Rahmen des Praktikums Computer Vision konnten wir erfolgreich einen ersten Prototypen unseres Smart Camera Operators erstellen, der autonom die Kameraführung von Videoaufnahmen übernimmt. Dabei konnten wir neben einigen Methoden der klassischen Computer Vision besonders mit Maschine Learning die Objekterkennung von Reitern und Pferden umsetzen und mit unterschiedlichen Verfahren beim Tracken experimentieren. Die erstellte Software wurde im Interesse der Kundengruppe benutzerfreundlich konzipiert, sodass diese, mithilfe einer guten Kamera, einen Ersatz für einen unerfahrenen Kameramann darstellt. Obwohl sich die Fähigkeiten von professionellen Kameraleuten enorm von unserem Prototypen abheben, konnten wir in der kurzen Zeit erhebliche Fortschritte erzielen und sind positiv eingestellt, dass sich mit einer Weiterentwicklung eine bereits deutlich professionellere, autonome Kameraführung realisieren lassen würde.

Für das geplante Einsatzgebiet für Reitsportturniere in NRW, sehen wir auch großes Potenzial, dass nach einem Praxistest ein weiträumiger Einsatz für regionale Turniere möglich und rentabel ist. Die bisherige Nutzung des Smart Camera Operator für Dressurreiten, kann in Zukunft mit wenig Aufwand für weitere Turnierarten wie Springreiten oder Pferderennen erweitert werden. Ebenfalls sehen wir großes Potenzial, dass diese Anwendung auch für Trainingsstunden der Reitsportler von Interesse ist, um die eigene Leistung ohne Kameramann reflektieren zu können.

# A. Videos

Alle Videos befinden sich in einem [Sciebo-Ordner](#), in dem die Ergebnis- und Siegerehrungsvideos getrennt zu finden sind. Dabei liegen jeweils das Originalvideo und das Ausgabevideo des Smart Camera Operators im selben Unterordner.

## A.1. Ergebnisse

Die Ergebnisvideos sind in [hier](#) zu finden und umfassen den folgenden Inhalt:

- [Video1](#): Einzelter Reiter Outdoor
- [Video2](#): Verdeckung des Reiterpaars
- [Video3](#): Zwei Reiter kreuzen erfolgreiches Tracking
- [Video4](#): Zwei Reiter kreuzen fehlerhaftes Tracking
- [Video5](#): Fokus auf jeweils eines von zwei Reiterpaaren
- [Video6](#): Detektion von Reiter und Pferd visualisiert durch eingezeichneten Bounding Boxen

## A.2. Siegerehrung

Die Videos für die Siegerehrung sind in [hier](#) zu finden und umfassen den folgenden Inhalt:

- [Video1](#): Einzelter Reiter Outdoor
- [Video2](#): Einzelter Reiter Outdoor2
- [Video3](#): Verdeckung des Reiterpaars
- [Video4](#): Zwei Reiter kreuzen
- [Video5](#): Fokus auf jeweils eines von zwei Reiterpaaren