Program pro analýzu zápasů stolního tenisu na turnajích TT Cup

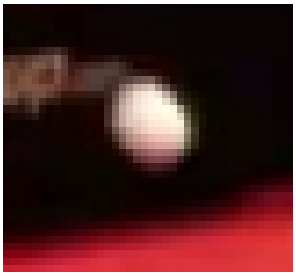
Ondřej Kadavý
ČVUT–FIT
kadavond@fit.cvut.cz

1. června 2025

Cílem mé prace je vytvořit program, který hráčům zjednoduší zpětnou analýzu zápasu a pořadatelům pomůže s procházením dlouhých turnajů.

1 Úvod

Během zpracovávání práce zjistil, že automatizace počítadla pouze za využití knihovny OpenCV bylo příliš optimistická, a to hlavně kvůli výraznému motion blur efektu ve vstupních videích a jejich relativně nízké kvalitě. Jakákoli segmentace na základě tvaru (například pomocí *Hough Circle Transform*, viz oficiální dokumentaci) nebo snaha segmentovat podle barvy nevedly k žádnému použitelnému výsledku.



Obrázek 1: Segmentovatelný míček (OpenCV)

Obrázek 2: Nesegmentovatelný míček (OpenCV)

Proto jsem se rozhodl využít předtrénovanou neuronovou síť YOLO, pro kterou jsem vytvořil databázi přibližně 2000 označených obrázků. Tyto obrázky jsem následně použil k natrénování detekčního modelu. Výsledná detekce již byla použitelná, nicméně i nadále docházelo k chybám – ať už k vynechání, nebo nesprávné detekci míčku. Pro účely automatizovaného počítadla, kde je vyžadováno výrazně přesnější sledování míčku, byl model jen mírně lepší než náhodný výběr.

Z tohoto důvodu jsem se nakonec rozhodl vytvořit program, který slouží k analýze jednotlivých zápasů turnaje. Lze jej využít ke zpětnému rozboru zápasu, což celý proces výrazně usnadňuje – dlouhé video, které obvykle obsahuje sedm zápasů, rozdělí na jednotlivá videa a zároveň zobrazí užitečné grafy, které ukazují, zda je potřeba zlepšit příjem, servis či dlouhé nebo krátké výměny.

2 Vstupní data

Vstupními daty jsou videa stažená z YouTube pomocí knihovny `yt_dlp`. Jedná se o videa z kanálu `TT Cup Czech 2`. Pro jednoduchost použití stačí vložit odkaz na video a dojde k automatickému stažení. Tedy pro využití stačí pouze připojení k internetu.

3 Postup

Celý postup jsem rozdělil do několika částí, které dohromady tvoří finální výsledek:

3.1 Rozdělení videa na jednotlivé zápasy

Tato část je relativně přímočará. Pomocí knihovny OpenCV procházím celé video a extrahuji klíčové snímky. Využívám funkci `cap.set(...)` k přeskočení určitých intervalů, v našem případě se jedná o 30 sekund, čímž se výrazně zkracuje doba zpracování — z přibližně jedné hodiny na jednotky minut.

Jednotlivé zápasy jsou poměrně snadno rozpoznatelné díky tomu, že před každým z nich se ve videu objeví obrazovka s logem, zatímco se hráči rozehrávají:

Obrázek 3: Obrázek oddělující jednotlivé zápasy

K identifikaci tohoto obrazu však nelze použít přesnou shodu, protože zřejmě dochází ke kompresi videa. Místo toho využívám metodu SSIM (Structural Similarity Index), z knihovny `scikit-image` a v tomto případě poskytuje velmi dobré výsledky.

Poté je video rozděleno na jednotlivé zápasy pomocí `FFmpeg`.Toto rozdělení výrazně ulehčuje práci v dalších částech.

3.2 Detekce skóre pomocí OCR

Pro detekci skóre jsem se rozhodl kombinovat segmentaci podle barvy s následnou detekcí pomocí Tesseract OCR. Tento přístup není zcela optimální, protože jsem testoval také EasyOCR, které využívá neuronové sítě a dosahuje výrazně lepších výsledků. Nicméně EasyOCR je výrazně pomalejší, a v současné podobě zabírá detekce přibližně třetinu délky videa, zatímco s EasyOCR běh trvá násobně déle. Proto jsem nakonec zůstal u Tesseract OCR.

Použití Tesseractu však přináší řadu problémů, například občasné chybné čtení obrázků, se kterými by EasyOCR nemělo větší potíže.

Obrázek 4: Ukázka chybného čtení pomocí OCR

Proto jsem při čtení skóre raději načetl téměř všechny možné vstupy a následně provedl jejich filtraci a doplnění. Pomocí sliding window kontroluji, zda v trojici po sobě jdoucích hodnot nedochází k propadu u prostřední hodnoty. Dále detekuji skoky ve skóre, filtruju duplicitní záznamy a doplňuji chybějící stavy, například skóre 0:0 na začátku setu. Pokud máme například zaznamenané skóre 1:0 a 3:0, automaticky doplním mezilehlý stav 2:0. Nakonec také dokončuji sety – pokud naposledy načtu skóre 10:2 a následně již máme záznamy z dalšího setu, doplním konečný výsledek 11:2. Pro spravnou funci vizualize je potřeba abychom měli všechny záznamy.

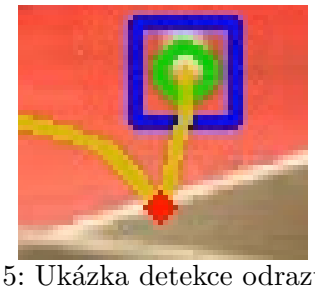
3.3 Sledování míčku pomocí YOLO

Jak již bylo zmíněno, pro sledování míčku využívám model YOLO s vlastním datasetem. Bohužel tato metoda také není bezchybná, a proto je potřeba provádět filtrace a doplňování výsledků podobně jako v předchozí části.

Vzhledem k tomu, že v každém snímku se vyskytuje vždy pouze jeden míček, je nutné v případě více detekcí ve stejném snímku vybrat tu nejpravděpodobnější. Při výběru se snažím vyvážit hodnotu confidence, kterou vrací model, a pozici míčku v prostoru. Nejprve zkoumám, zda se některá z detekcí nachází v blízkosti pozice míčku z předchozího snímku. Pokud žádná taková detekce neexistuje nebo nejsou k dispozici předchozí záznamy, vyberu detekci s nejvyšší confidence.

Poté provádím filtraci výsledků, kdy kontroluji, aby skoky v pozicích míčku nebyly příliš velké. Dále lineárně doplňuji chybějící pozice míčku, avšak pouze v případě, že mezera bez detekce míčku nepřesahuje 4 snímky.

Následně provádím detekci odrazu míčku od desky stolu. Odraz je zaznamenán pouze tehdy, pokud se míček nachází v prostoru stolu. K detekci používám také sliding window, kdy vždy beru tři po sobě jdoucí pozice a kontroluji, zda je prostřední pozice nižší než obě okolní. Tento vzor tvoří tvar „V“ v trajektorii míčku. Dále aplikuji filtr na změnu, aby nedocházelo k falešným detekcím způsobeným šumem.

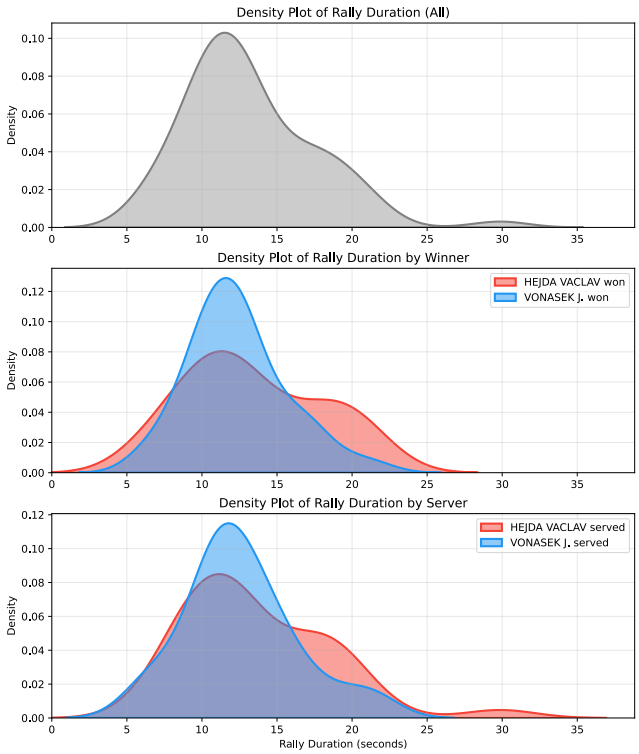

Obrázek 5: Ukázka detekce odrazu a míčku

3.4 Vizualizace výsledků

Pro vizualizaci jsem primárně využíval knihovny `matplotlib` a `seaborn`, především z důvodu mých předchozích zkušeností s těmito knihovnami. Vizualizace lze rozdělit do dvou částí: vizualizace dat získaných z OCR a vizualizace dat z části detekce pomocí YOLO.

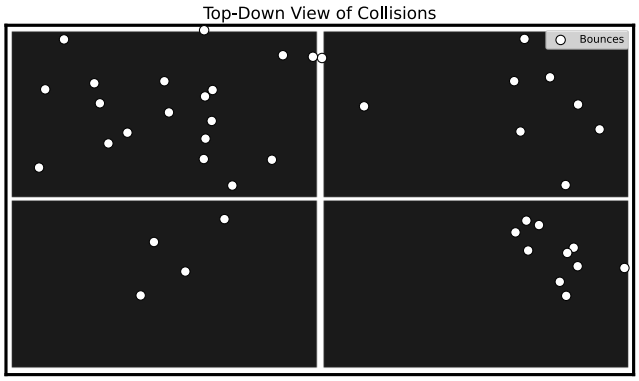
Data získaná z OCR jsem převedl do objektu `pandas.DataFrame` a doplnil o informace jako například, kdo podával, kdo vyhrál výměnu a jak dlouho

výměna trvala. Poté jsem tyto údaje dále zpracoval a vizualizoval pomocí vhodných grafů.



Obrázek 6: Ukázka grafů, který ukazuje distribucí

Vizualizace dat získaných pomocí YOLO byla o něco složitější, protože stůl je na videu snímán v zajímavé perspektivě. Nejprve bylo nutné převést detekované pozice míčku do ortogonálního (top-down) pohledu pomocí funkcí `cv2.findHomography()` a `cv2.perspectiveTransform()` z knihovny OpenCV. Teprve poté bylo možné výsledky vhodně zobrazit, například jako náhled stolu shora nebo pomocí heatmapy, která ukazuje frekvenci dopadů míčku.



Obrázek 7: Ukázka grafů, který ukazuje distribucí

3.5 UI pomocí streamlitu

Původně jsem předpokládal, že celý program nebude mít žádné uživatelské rozhraní, nebo maximálně jednoduché ovládání přes Jupyter Notebook. Po cvičení, kde byl představen Streamlit, jsem ale zjistil, že přesně odpovídá mým požadavkům. Jeho hlavní výhodou je velmi snadné zprovoznění ve srovnání například s PyQt. Na druhou stranu, pokud by bylo potřeba implementovat složitější logiku uživatelského rozhraní, mohl by se stát obtížně udržitelným řešením. Pro mé účely však plně postačuje. Největší komplikací bylo zprovoznění vícestránkové aplikace s využitím session state.

4 Testy

Testy jsem se snažil stavět na reálných datech, která jsem vytvořil z videí ze skutečných zápasů. V testování jsem se zaměřil především na klíčové funkce systému. Naopak jsem netestoval funkce pro generování grafů, protože mi to v tomto kontextu nepřišlo smysluplné.

5 Codestyle

Snažil jsem se co nejvíce dodržovat doporučení PEP8. Jedinou výjimkou bylo pravidlo `E1101`, které způsobovalo problémy při práci s knihovnou `cv2`. Dále jsem musel vypnout hlášení `no-name-in-module` pomocí `# pylint: disable=no-name-in-module`, protože způsobovalo chyby při importu `ssim`.

6 Výsledky

Výsledkem práce jsou finální grafy a statistiky daného zápasu. Dále také video daného zápasu a vizualizace trajektorie míčku a dopadů na stůl pro segment videa.

Důležité je také zmínit místa, kde mohou výsledky působit zvláštně. Největší problém vzniká, když OCR kontinuálně vrací nesprávné hodnoty – například pokud se v řadě 20 záznamů místo skóre setů 0:0 načte 2:2, algoritmy na filtraci pak nefungují správně a výsledné grafy jsou zkreslené. Dále jsem narazil na problém posunu kamery, který sice OCR neovlivnil, ale způsobil posun trackingu a celkových vizualizací. Kalibrace byla provedena na video YouTube video, a k datu 31. 5. 2025 je nastavení stále stejné. Více v další sekci.

Důležité je věnovat pozornost době běhu jednotlivých částí programu. Detekce pomocí YOLO může být velmi časově náročná, proto doporučuji pracovat s kratšími intervaly. Také střih videa a čtení pomocí OCR mohou trvat několik minut, v některých případech i desítky minut.

Dále je potřeba dávat pozor na UI ve Streamlitu, protože se často stává, že se některé části neudrží nebo dojde ke změnám, které způsobí chyby v kódu a UI přestane správně fungovat.

7 Využití LLM

Největší pomocí, kterou mi LLM (konkrétně ChatGPT) poskytl, bylo generování docstringů. U většiny důležitých funkcí jsem si je nechal vygenerovat, u několika klíčových jsem je však psal ručně. V některých případech mě k využití generování donutilo dodržování standardu PEP8.

ChatGPT jsem dále využíval při psaní souboru README a komentářů v kódu – zpravidla jsem napsal anglický text vlastními slovy a následně se dotazoval na gramatickou správnost, protože chci, aby výsledný text působil lépe. Texty v češtině, včetně tohoto, jsou výhradně mé vlastní, což se může projevit drobnými překlepy, i když jsem je po sobě kontroloval.

Jako užitečný nástroj jsem ChatGPT používal také při rychlém vyhledávání v dokumentaci nebo při objevování nových funkcí a metod. Například takto jsem narazil na funkci cv2.set(...), když jsem řešil problém s extrémně dlouhým během původní verze programu.

8 Závěr

Tento program sám aktivně využívám, především pro účely sestřihu turnajů. Ruční proklikávání jednotlivých videí je časově náročné, a pokud si chci video uchovat pro pozdější analýzu, je velmi praktické mít ho již připravené ve formě jednotlivých zápasů. Grafy a vizualizace navíc poskytují cenný přehled o průběhu utkání.

Pro další zlepšení by bylo vhodné implementovat detekci okrajů stolu, která by zajistila konzistentní zarovnání scény a tím zvýšila přesnost vizualizace.

Také by stálo za zvážení vytvořit vlastní dataset pro OCR a natrénovat Tesseract na specifický scoreboardu. To by pravděpodobně výrazně zvýšilo kvalitu čtení skóre.

V případě analýzy pomocí YOLO by bylo ideální vytvořit přibližně 5000 anotovaných snímků a využít výkonnější model než YOLOv8-nano. Bohužel, nemám k dispozici GPU (CUDA), a dostupný čas na Google Colabu jsem již vyčerpal.

Po implementaci těchto vylepšení by aplikace mohla bez problémů fungovat i pro všechny zápasy série TT Cup, nejen pro kanál `TT Cup Czech 2`.

Zdroje

[1] OpenCV documentation. online, 2025. [cit. 2025–5–31] `https://docs.opencv.org/4.x/index.html`.

[2] TT Cup Czech 2. online, 2025. [cit. 2025–5–31] `https://www.youtube.com/@ttcupczech2`.

[3] Magda Friedjungová. BI - VIZ. online, 2025. [cit. 2025–5–31] `https://courses.fit.cvut.cz/BI-VIZ/index.html`.

[4] Marcel Jiřina. BI - SVZ. online, 2025. [cit. 2025–5–31] `https://courses.fit.cvut.cz/BI-SVZ/index.html`.

[5] Murtaza's Workshop - Robotics and AI. I Tried Adding Computer Vision to Table Tennis. online video, 2025. [cit. 2025-05-31]`https://www.youtube.com/watch?v=WLCqhg26Gxc&t`.

[6] Sergey Lukianchenko. Table Tennis Detection GitHub Repository, 2025. [cit. 2025-05-31] `https://github.com/lukianchenko/table-tennis-detection?tab=readme-ov-file`.

[7] Streamlit Inc. Streamlit Documentation, 2025. [cit. 2025-05-31]`https://docs.streamlit.io/`.

[8] Tesseract OCR contributors. Tesseract OCR GitHub Repository, 2025. [cit. 2025-05-31] `https://github.com/tesseract-ocr/tesseract`.

[9] Ultralytics. YOLO documentation. online, 2025. [cit. 2025–5–31] `https://docs.ultralytics.com/`.

[10] T. Zou, J. Wei, B. Yu, and et al. Fast moving table tennis ball tracking algorithm based on graph neural network. *Scientific Reports*, 14:29320, 2024. [cit. 2025–05–31] `https://doi.org/10.1038/s41598-024-80056-3`.