GYMNASIUM JANA KEPLERA

Parléřova 2/118, 169 00 Praha 6

SimonPose

Maturitní práce

Autor: Šimon Brávek

Třída: 4.B

Školní rok: 2025/2026

Předmět: Informatika

Vedoucí práce: Jiří Matas

Praha, 2026

Gymnázium Jana Keplera

Kabinet informatiky

ZADÁNÍ MATURITNÍ PRÁCE

- *Student:* Šimon Brávek
- *Třída:* 4.B
- *Školní rok:* 2025 / 2026
- *Vedoucí práce:* Jiří Matas
- *Název práce:* Vylepšení odhadu pózy člověka z jednoho vstupního snímku
- URL repozitáře: https://github.com/simonbravek/pose-detection-project

---

Pokyny pro vypracování:

Cílem práce je navrhnout, implementovat a experimentálně ověřit metodu pro **přesnější odhad lidské pózy** z jednoho RGB snímku.

Student využije již existující model **DensePose** pro získání hustého korespondenčního mapování povrchu těla a naváže na něj vlastní optimalizační nebo post-processingovou metodu s cílem:

- snížit chyby v odhadu pozice a orientace klíčových kloubů,
- případně zvýšit robustnost vůči šumu a odlišnostem v pozici či osvětlení vstupního snímku.

Součástí práce je:

- rešerše současných přístupů k odhadu pózy,
- implementace vlastního řešení v prostředí Python (TensorFlow),
- návrh metrik pro hodnocení přesnosti a srovnání s výchozím stavem,
- experimentální vyhodnocení na vybraném datasetu,
- prezentace dosažených výsledků a postupu práce formou závěrečné zprávy a ukázky funkčního systému.

Výstupy práce:

- zdokumentovaný software v repozitáři,
- technická zpráva shrnující postup, metodiku a dosažené výsledky,
- prezentace práce a její obhajoba.

Podpis vedoucího práce:

Prohlášení

Prohlašuji, že jsem svou práci vypracoval samostatně a použil jsem pouze prameny a literaturu uvedené v seznamu bibliografických záznamů. Nemám žádné námitky proti zpřístupňování této práce v souladu se zákonem č. 121/2000 Sb. o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon) ve znění pozdějších předpisů.

V Praze dne 2. března 2026 Šimon Brávek

Deklarace užití nástrojů umělé inteligence

V rámci přípravy této maturitní práce byly využity nástroje generativní umělé inteligence (AI). Ta byla použita k řešení dílčích úkolů, zejména pro ladění a optimalizaci zdrojového kódu a stylistickou úpravu textu. Autor práce provedl kritické zhodnocení všech výstupů a nese plnou odpovědnost za konečný obsah a správnost textu.

Poděkování

Děkuji Jiřímu Matasovi za skvělou příležitost strávit měsíc v létě na katedře strojového učení a být součástí soudobého výzkumu a za jeho skvělé nápady v tomto oboru. Bezpochyby tato zkušenost formovala mé rozhodnutí studovat obor AI ma MFF UK a propůjčila mi vášeň k nejnovějším technologiím. Děkuji také Matěji Suchánkovi jako člověku, co mě provedl tou nejtěžší asimilací s prácí s grafickými kartami a modely jež jsem používal. Děkuji také lidem z katedry, jež ke mě byli přátelští a v neposlední řadě Igoru Vujovičovi za rozvíjení informatiky na našem gymnáziu i přez mnohé překážky.

Abstrakt

Tato maturitní práce se zabývá odhadem lidské pózy (human pose estimation) v reálných scénách (in-the-wild) a možnostmi, jak zpřesnit monokulární 3D rekonstrukci člověka z běžného 2D snímku. Vychází z toho, že současné modely dosahují výborných výsledků na laboratorních datasetech, ale v praxi je limitují zákryty, perspektivní zkreslení, vizuální šum a propletení více těl. Práce proto kombinuje DensePose, který poskytuje hustou korespondenci pixelů na povrch těla, s parametrickým 3D modelem SMPL, jenž vynucuje globálně konzistentní a anatomicky realistickou geometrii lidského těla.

Zvolená metodika je explorativní a řídí se principem Fail-Fast: rychle formulovat hypotézu, ověřit ji prototypem a analyzovat i neúspěšné výsledky. Jsou navrženy tři varianty fittingu (pasování) modelu SMPL do signálu z DensePose: „Embedding fitter“ s explicitním ošetřením viditelnosti, stabilnější „Euklidovský fitter“ měřící chybu v obrazové rovině a prototyp „Precizní fitter“, který odhaduje důvěryhodnost korespondencí podle jejich prostorové konzistence. Experimenty na COCO DensePose (minival) ukazují, že embeddingový přístup je výpočetně náročný a nestabilní. Naopak euklidovská formulace umožňuje rychle iterovat experimenty a přibližně u třetiny zkoušených snímků dosahuje 2D překryvu srovnatelného s původní detekcí; výsledkem je navíc explicitní 3D model těla. Součástí výstupu práce je také otevřený repozitář SimonPose s implementacemi prototypů a technickou dokumentací, který může sloužit jako základ pro další zvyšování robustnosti fittingu a navazující výzkum v oblasti počítačového vidění.

Klíčová slova

počítačové vidění, odhad lidské pozice, DensePose, SMPL/SMPL-X,
rekonstrukce z 2D do 3D, hluboké učení, vylepšení modelu

Abstract

This thesis addresses human pose estimation in real-world (in-the-wild) imagery and investigates ways to improve monocular 3D human reconstruction from a single 2D image. While state-of-the-art models perform well on controlled benchmarks, their accuracy degrades under occlusions, perspective distortion, visual noise, and crowded scenes. To mitigate these issues, this work combines DensePose, which provides dense pixel-to-surface correspondences, with the parametric 3D body model SMPL, which enforces globally consistent and anatomically plausible human geometry.

The methodology is exploratory and follows a Fail-Fast approach: quickly formulate a hypothesis, validate it with a prototype, and analyze failures as well as successes. Three SMPL fitting variants are proposed for leveraging the DensePose signal: an „Embedding fitter“ with explicit visibility handling, a more stable „Euclidean fitter“ that measures error in the image plane, and a „Precision fitter“ prototype that estimates correspondence reliability from spatial consistency. Experiments on COCO DensePose (minival) show that the embedding-based approach is computationally expensive and unstable. In contrast, the Euclidean formulation enables rapid iteration and achieves 2D overlap comparable to the original detection on roughly one third of the evaluated images, while additionally producing an explicit 3D body model. The thesis also delivers the open-source SimonPose repository, including prototype implementations and technical documentation, providing a foundation for improving fitting robustness and for follow-up research in computer vision.

Keywords

computer-vision, human-pose-estimation, DensePose, SMPL/SMPL-X,
2D-to-3D reconstruction, deep-learning, model improvement

Obsah

Slovník pojmů

model naučená neuronová síť, podrobněji v teoretické části

segmentace (segmentation) rozdělení obrázku na skupiny pixelů, typicky dle objektů, které popisují. v kontextu této práce tím referuji ke skupině pixelů patřící detekovanému lidskému tělu.

bounding box je nejmenší obdélník obsahující celou segmentaci nějakého objektu, typicky se značí pomocí dvou bodů (levého horního rohu a pravého dolního rohu)

detekce (detection) je objekt zaznamenaný modelem, typicky pomocí bounding boxu

maska (mask) výsledek segmentace, je to černo-bílý obraz stejné velikosti jako originál překrývající segmentaci objektu, může být i binární maskou, kde k pixely patřící k objektu mají hodnotu 1 a ostatní hodnoty 0

embedding je vektorové pole zachycující význam obrázku, více v teoretické části

1. Úvod

1.1 Analýza postavy v digitálním světě

V současné informatice a počítačovém vidění (Computer Vision) představuje automatické porozumění lidského pohybu jednu z největších výzev. Nejde již o prostou detekci zda se v obrázku nachází člověk, či kolik jich je, ale o snahu přenést lidskou biomechaniku do digitálního prostoru. Schopnost přesně interpretovat lidskou pózu z běžného 2D obrazu (např. z mobilního telefonu) otevírá dveře aplikacím, které byly dříve nemyslitelné bez drahých studiových systémů pro snímání pohybu (Motion Capture).

Tyto technologie se dnes nachází již v mnoha podobách a můžou tak pomoci v celé řadě aplikací. Dnes již můžeme najít rozpoznání člověka a obličeje v mobilní aplikaci galerie, kde napomáhá chytrému třízení fotek nebo v bezpečnostních systémech pro detekci vetřelce. Detekce konkrétní pozice, tedy kde se nachází jednotlivé klouby, může zase pomoci zastavit automatický vozík před kolizí s člověkem nebo zavolat pomoc pokud uvidí nehybně ležícího chodce.

Kromě přesnosti těchto metod se vědci čím dál více zaměřují také na rychlost. Mnohé aplikace totiž potřebují živou analýzu videa a to často i z několika kamer najednou. Tak tomu je u autonomního řízení automobilů, které se ukazuje být velkým tahounem tohoto odvětví.

Právě rozmanitost této disciplíny, její užitečnost a také možnost setkat se s ní do hloubky na FEL ČVUT mě vedlo k její volbě jako tématu mé maturitní práce.

1.2 Problematika

Současné SOTA (State-of-the-Art) modely sice dosahují vynikajících výsledků na laboratorních datasetech, ale často selhávají v reálných podmínkách (in-the-wild). Mezi kritické faktory patří:

- **Zákryty (Occlusions):** Části těla zakryté předměty nebo jinými částmi těla.
- **Perspektivní zkreslení:** Extrémní úhly kamery, které deformují vizuální proporce těla.
- **Vizuální šum:** Volné oblečení nebo špatné světelné podmínky.
- **Propletení více těl:** Situace v davu, kde není jasné, komu patří jaká část těla a jak je správně rozdělit.

Právě kombinace modelu SMPL s technologií DensePose (která mapuje pixely přímo na povrch těla) nabízí unikátní cestu, jak tyto problémy řešit skrze hustou korespondenci dat.

1.3 Cíl práce

Cílem této maturitní práce není vytvořit nový, revoluční model, který by překonal stávající vědecké rekordy. Ambicí je metodický průzkum možností, jak stávající proces fittingu (pasování) modelu SMPL do dat z DensePose zpřesnit a učinit jej odolnějším (robustnějším).

Zvolená metodika se opírá o princip Fail-Fast:

- Rychlá formulace hypotéz o vylepšení optimalizačního procesu.
- Implementace prototypů a jejich testování na hraničních případech.
- Analýza selhání jakožto hlavního zdroje poznání.

Výsledkem práce je ucelený přehled vyzkoušených metod, jejich kritické zhodnocení a dokumentace slepých i perspektivních uliček, které mohou sloužit jako inspirace pro další vývoj v oblasti monokulární 3D rekonstrukce člověka.

1.4 Struktura práce

Teoretická část shrnuje klíčové pojmy a principy použitých metod (DensePose a parametrický model SMPL) a motivuje, proč jejich kombinace dává smysl pro robustnější odhad pózy v reálných scénách. Praktická část popisuje tři zkoušené varianty fittingu SMPL do signálu z DensePose, včetně implementačních kompromisů a zhodnocení. Technická dokumentace slouží jako praktický návod k instalaci a reprodukci experimentů v repozitáři SimonPose. Závěr stručně shrnuje dosažené výsledky, limity a možné směry pokračování.

2. Teoretická část

2.1 Základní teorie

2.1.1 Co je to model

V této práci se často budu věnovat programům, které využívají metodu strojového učení k detekci a odhadu pozice člověka v obrázku, který nazýváme model. Tento program má mnoho parametrů na základě nichž přetváří vstup na výstup. Nejdříve projde procesem, které nazýváme trénování. Během toho se jako vstup použije obrázek a program k němu vygeneruje souřadnice, kde odhaduje klouby, na základě prvního nastavení parametrů, neboli inicializace. Inicializace bývá velmi složitá a často je předmětem celých studií [1] a nebudeme se jí věnovat v této práci. .Tento výsledek se pak porovná s ukázkovým příkladem pro tento obrázek, tedy pro anotaci, která byla předtím připravena člověkem.

Dále zvolíme metodu pro výpočet chyby výstupu (loss). Proces generování výstupu opakujeme mnohokrát pro velké množství obrázků a anotací toho. Skupině těchto dat říkáme trénovací dataset. Je nutné mít velké množtví obrázků a anotací ještě před začátkem trénování. Ty pochází v drtivé většině případů od lidských anotátorů a jsou tak velmi drahé. Navíc jsou předmětem lidské chybovosti. Součtem, nebo jinou metodou tak spočítáme ztrátovou funkci (loss funcion) jako metriku chybovosti odhadu našeho modelu od anotací

2.1.2 Druhy detekcí

Když v počítačovém vidění mluvím o „detekci“, nemusí tím být myšlen pouze rámeček kolem objektu. Model může výstupem popsat jak samotnou polohu objektu, tak i jeho tvar, strukturu nebo konkrétní body. Pro úlohy, které souvisí s člověkem (a se kterými se v této práci setkávám), jsou nejčastější zejména tyto typy detekce:

- **Bounding box:** nejmenší obdélník, který postavu ohraničí. Je výpočetně levný a často slouží jako první krok, který zúží oblast zájmu pro další analýzu.
- **Instance segmentace (maska):** pixelová reprezentace siluety člověka pro každou osobu zvlášť. Oproti rámečku zachycuje přesnější tvar a lépe funguje v přítomnosti zákrytů.
- **2D klíčové body (keypoints):** sada anatomických bodů (např. ramena, lokty, kolena), které tvoří „kostru“ pózy. Jde o řídkou reprezentaci, která je rychlá a interpretovatelná, ale ztrácí informaci o objemu těla.
- **Hustá korespondence (DensePose):** přiřazení pixelů na postavě ke kanonickému povrchu těla. Tento výstup poskytuje výrazně bohatší signál než samotné klíčové body a tvoří základ metod, které v této práci zkouším.

V praxi bývá detekce doplněna i skóre důvěry (confidence), podle kterého lze výsledky filtrovat. Volba typu detekce pak přímo určuje, jaké metriky kvality dávají smysl (např. IoU pro masky, nebo průměrnou chybu bodů pro keypoints) a jaké další kroky jsou vůbec možné.

2.1.3 Odhad 2D a jeho limitace

Tradiční metody odhadu pózy se po léta soustředily na tzv. sparse keypoints – detekci klíčových bodů, jako jsou lokty, kolena či ramena – které detekují jako dvojici souřadnic v daném obrázku. Přestože jsou tyto modely (např. OpenPose) rychlé a efektivní, trpí zásadním nedostatkem: ztrátou prostorové informace a tělesného objemu. Zatímco mají perfektní výsledky v laboratorních podmínkách, selhávají v reálných situacích (in-the-wild), kde jim 2D obrázek neposkytuje dost jasné informace pro detekci pozice. Mezi faktory ovlivňující přesnost výstupu patří:

- **Zákryty (Occlusions):** Části těla zakryté předměty nebo jinými částmi těla.
- **Perspektivní zkreslení:** Extrémní úhly kamery, které deformují vizuální proporce těla.
- **Vizuální šum:** Volné oblečení nebo špatné světelné podmínky.
- **Propletení více těl:** Situace v davu, kde není jasné, komu patří jaká část těla a jak je správně rozdělit.

2.1.4 Od 2D bodů k 3D objemu

Řešení těchto složitých případu významně napomáhá představa 3D modelu těla a jeho pozice na dané scéně. Vezmeme-li situaci člověka skákajícího na lyžích zespudu, jeho tělesné proporce budou zcela nestandardní (malá hlava, velké nohy, spousta zakrytých částí těla). Pokud si ale představím model lidského těla a promítnu ho do obrázku, pak jsem značně omezen a najít správné orientace končetin se značně zjednoduší. Představa těla ve 2D mi naopak dovoluje zvažovat pozice, jež by byli lidské tělo zcela nepřirozené, či anatomicky nemožné. Modely, které spolu s pozicí kloubů odhadují také 3D orientaci těla se díky tomu stávají lepší v samotném odhadování pozice kloubů.

Samotné rozpoznávání 2D pozice se dostává ke svým limitům také proto, že jejich přesnost se blíží datům na kterých jsou trénovány. [2]. Další k rozpoznání 3D pozice těla je zle

2.2 Skinned Multi-Person Linear Model (SMPL)

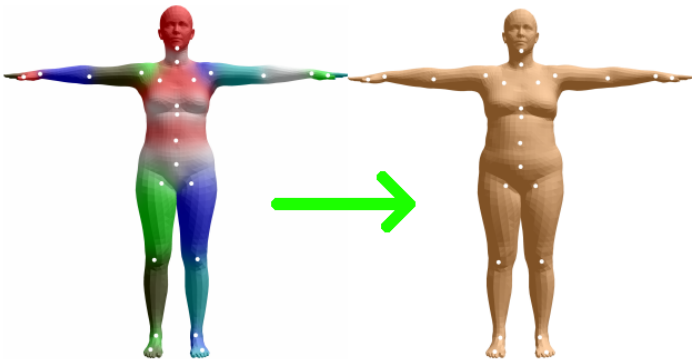
Přechod od 2D chápání člověka k pochopení objemové struktury lidského těla si vyžaduje zcela nové nástroje. Jedním z nich je způsob jak modelovat lidské tělo v prostoru. Abychom mohli tělo vyrendrovat na obrazovku pomocí standardních postupů, musíme povrch těla zapsat jako množinu bodů a zapamatovat si všechny možné trojice bodů tak, aby nám vzikla síťovina (mesh) reprezentující povrch 3D tělesa.

Namodelovat realistické lidcké tělo na základě bodů je velmi náročná disciplína. My bychom takové tělo chtěli modelovat automaticky, v reálném čase a s tělesnými proporcemi a pozicí, jakou si zadáme. Proto vznikl SMPL model, který na základě parametrů β a θ vytvoří síťovinu libovolného člověka.

2.2.1 Výpočet

Celý model těla vzniká tak, že se začne se souřadnicemi bodů kanonického těla $\bar{T}$ v klidu (tedy univerzálního zvoleného těla, které zobrazuje průměrného člověka ve všech proporcích). Ke každému bodu se následně přičte posunutí na základě parametru tělesných proporcí β příspěvkem $B_s(\beta)$.

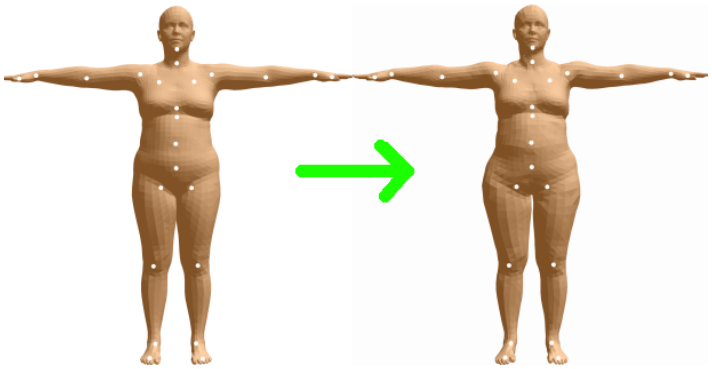
Tak vznikne T-pose správných tělesných proporcí. Také vypočteme počáteční pozici kloubů $J(\beta)$ na základě tělesných proporcí. Viz Obrázek 2.1.



(a) $\bar{\mathbf{T}}, \mathcal{W}$ **(b)** $\bar{\mathbf{T}} + B_S(\vec{\beta}), J(\vec{\beta})$

Obrázek 2.1: SMPL – stage 1

Při pohybu se naše tkáně napínají a ohýbají a s tím musíme počítat i u počítačového modelu člověka. Dále tedy upravíme proporce těla tak, aby odpovídali po pozici do které chceme model dostat tím, že k modelu přidáme příspěvek $B_P(\theta)$ a vznikne tak pozice $T_p(\beta, \theta)$. Viz Obrázek 2.2.



(b) $\bar{\mathbf{T}} + B_S(\vec{\beta}), J(\vec{\beta})$ **(c)** $T_P(\vec{\beta}, \vec{\theta}) = \bar{\mathbf{T}} + B_S(\vec{\beta}) + B_P(\vec{\theta})$

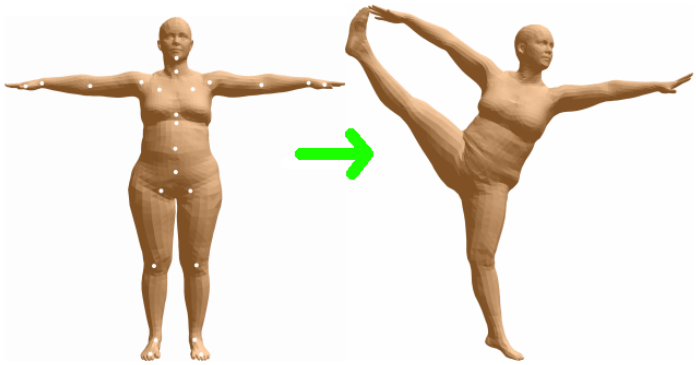
Obrázek 2.2: SMPL – stage 2

Nakonec se vše poskládá dohromady pomocí funkce $W(\cdot)$ ještě s konečnou pozicí končetin θ a maticí $\mathcal{W}$. Tato matice spolu s $B_s(\beta)$ a $B_p(\theta)$ jsou natrénované hodnoty na tisících lidských skenů. Viz Obrázek 2.3.

2.2.2 Výhody

Tento model má výhodu v tom, že parametry θ a β usměrní výslednek tak, aby bylo velmi těžké zdeformovat obrázek do nelidských proporcí. Zároveň však dokáže popsat celou řadu lidských těl. Je díky tomu vhodná pro

Tato knihovna je pro komerční účely zpoplatněná, ale pro vědecké účely je zdarma a stačí se registrovat na `https://smpl.is.tue.mpg.de/index.html`. Díky tomu se z ní stal standard v oblasti počítačového vidění.



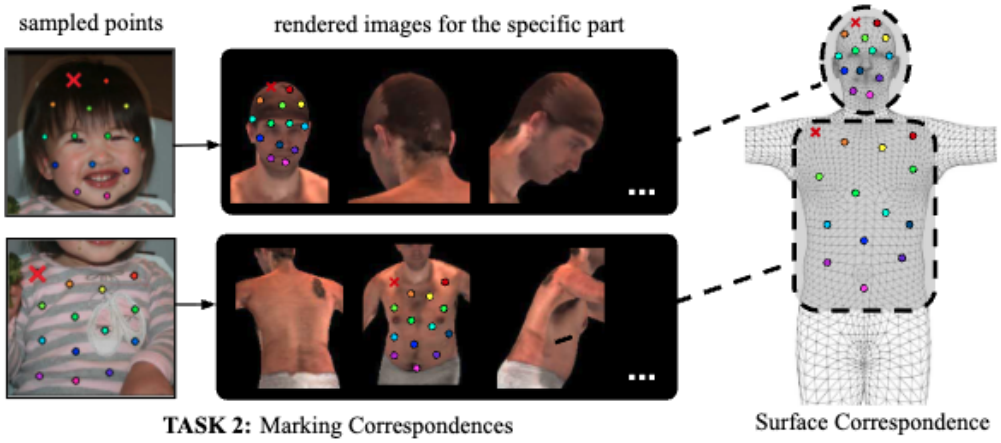
(c) $T_P(\vec{\beta},\vec{\theta}) = \bar{\mathbf{T}} + B_S(\vec{\beta}) + B_P(\vec{\theta})$ (d) $W(T_P(\vec{\beta},\vec{\theta}), J(\vec{\beta}), \vec{\theta}, \mathcal{W})$

Obrázek 2.3: SMPL – stage 3

2.3 DensePose

DensePose je metoda z oblasti počítačového vidění, která z obyčejného 2D obrázku odhaduje **hustou korespondenci** mezi pixely a povrchem lidského těla. Na rozdíl od klasického odhadu pózy pomocí klíčových bodů (např. lokty a kolena) se zde nepracuje s několika desítkami bodů, ale s tisíci pixelů, které lze přiřadit ke konkrétním místům na těle. Prakticky to znamená, že pro každý pixel patřící člověku dokáže DensePose určit, *kam na 3D povrch těla by tento pixel patřil*, pokud bychom měli k dispozici standardizovanou šablonu těla.

Jádrem celé myšlenky je kanonická reprezentace těla (tedy povrch „průměrného“ člověka v jednotných souřadnicích) a síť, která se učí předpovídat mapování z obrazu do této kanonické reprezentace. DensePose je typicky používán jako nadstavba detekce osob: nejprve se nalezne ohraničující rámeček (bounding box) postavy a teprve v jeho rámci se provede husté mapování pixelů na povrch těla. Díky tomu se výpočet soustředí na relevantní část obrazu a je možné pracovat s výstupem po jednotlivých lidech.



Obrázek 2.4: DensePose – DenseAnnoFigue, Zobrazuje korespondenci 2D pixelu s vybraným bodem na kanonické 3D síti lidského těla.

2.3.1 DensePose v1: mapování pomocí (I, U, V)

Původní verze DensePose (v této práci ji budu označovat jako DensePose v1) používá parametrizaci povrchu těla pomocí několika tělesných „chartů“ (segmentů). Výstupem je pro každý pixel na postavě trojice (I, U, V):

- **I:** index tělesného segmentu (např. levou paži, trup apod.),
- **U, V:** souřadnice v rámci 2D parametrizace daného segmentu.

Tento výstup je vhodný například pro přenos textur nebo pro vizualizaci mapování povrchu, ale pro následný fitting parametrického modelu (např. SMPL) je potřeba navíc řešit převod z (I, U, V) do konkrétních bodů nebo vrcholů (vertices) na síťovině. To je možné, ale v praxi to přidává další vrstvu komplikací, protože optimalizační algoritmus pak nepracuje přímo s jednoznačnou korespondencí na SMPL mřížce.

2.3.2 DensePose v2: Continuous Surface Embeddings (CSE)

Novější varianta, kterou v této práci používám (DensePose v2), přechází od explicitních (I, U, V) souřadnic k tzv. Continuous Surface Embeddings (CSE). Místo toho, aby síť přímo vracela parametrické souřadnice na těle, vrací pro každý pixel vektor embeddingu (tedy bod v naučeném vektorovém prostoru), který reprezentuje odpovídající místo na povrchu těla. Současně vrací i hrubou segmentaci (coarse segmentation), která říká, které pixely v daném rámečku vůbec patří postavě.

Z praktického hlediska tak DensePose v2 produkuje dvě hlavní struktury:

- **Masku** S o rozměrech $H \times W$ (binární nebo vícetřídní), která určuje pixely patřící tělu.
- **Matici vektorů** E o rozměrech $H \times W \times D$, kde D je dimenze embeddingu, a každý pixel tak nese svůj vektor $\mathbf{e} \in \mathbb{R}^D$. DensePose byl natrénován tak, aby vektory korespondující s blízkými body byli podobné (skalární součin je maximální) a pro vzdálené body rozdílné (skalární součin je minimální).

Tyto mapy jsou definované v souřadnicích rámečku detekované osoby a lze je přepočítat na libovolné rozlišení (typicky interpolací na nové H a W). V praxi je to důležité, protože výstup se dá škálovat podle toho, zda preferuji rychlost (nižší rozlišení) nebo přesnost a počet korespondenčních bodů (vyšší rozlišení). Pro účely optimalizace je pak možné z embeddingů odvodit i jednoznačný index vrcholu na SMPL síťovině: pro každý pixel se vybere ten vrchol, jehož předpočítaný embedding je vektorově nejbližší (typicky podle kosinové podobnosti). Výsledkem je tedy mřížka indexů, která mapuje pixely přímo na vrcholy SMPL (např. 6890 vrcholů), a tu lze přímo použít v loss funkci.

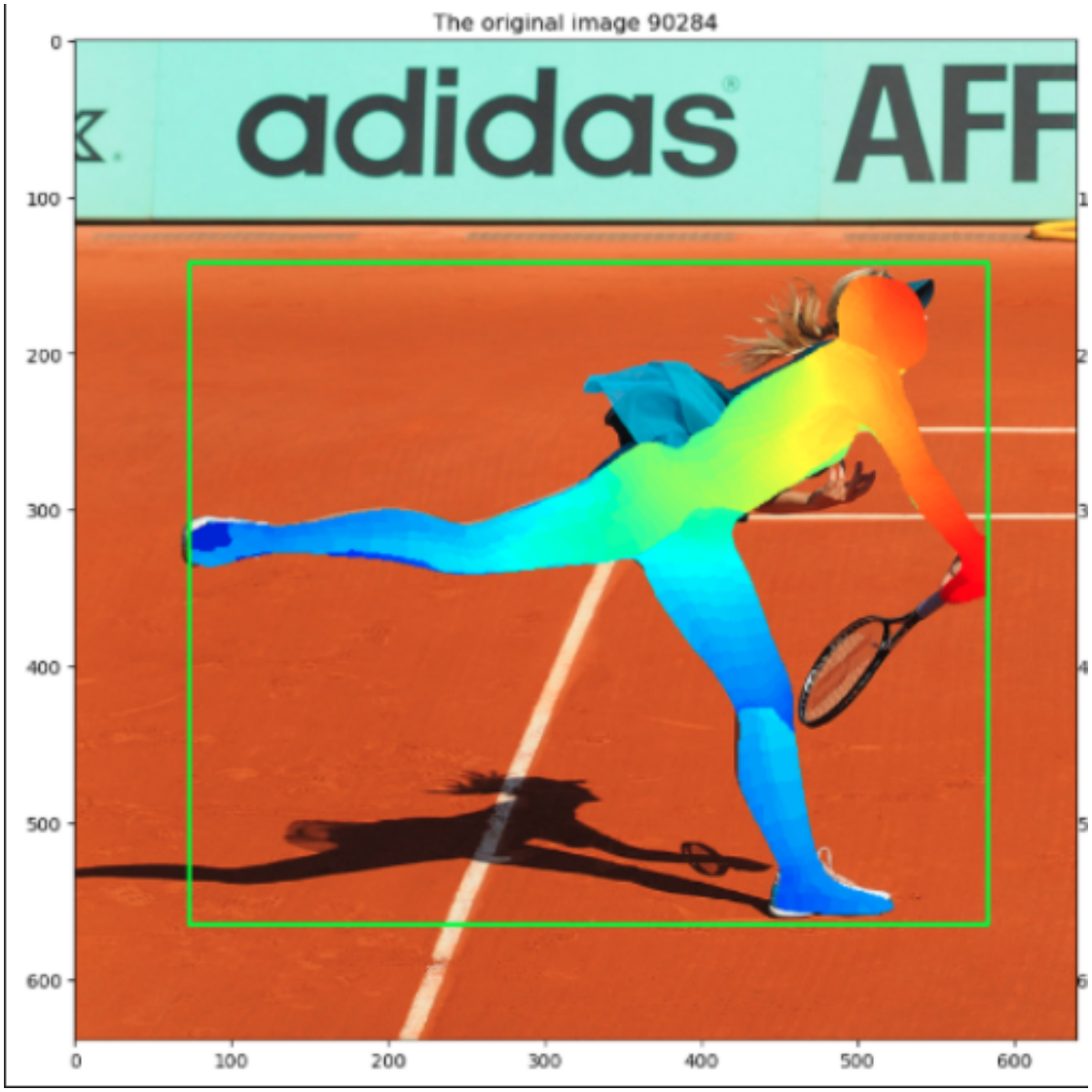
DensePose v2 jsem zvolil z několika důvodů. Především mě překvapila jeho přesnost i v podmínkách mimo laboratorní datasety a zároveň jde o open-source řešení postavené nad ekosystémem Detectron2, které je dobře integrovatelné do vlastního pipeline. Zároveň je férové říct, že DensePose jako projekt dnes působí spíše jako *vědci trochu zapomenutá* technologie: nové práce často preferují holistické modely, které rovnou odhadují parametry těla nebo celé 3D reprezentace. DensePose však zůstává velmi vhodný pro navazující práci právě kvůli tomu, jaký typ výstupu poskytuje. Moderní holistické modely mívají výstup „hotový“ (např. klouby nebo SMPL parametry), ale neposkytují jednoduchý způsob, jak si vyžádat hustou, škálovatelnou korespondenci pixelů na povrch těla, kterou lze přímo zapojit do vlastní optimalizace.

2.3.3 Silné a slabé stránky

Největší výhodou DensePose je, že poskytuje hustý signál: i když část těla není jasně vidět, zbytek povrchu často dává dostatek informací pro smysluplné omezení 3D řešení. Tento typ výstupu je navíc přirozeně kompatibilní s fittingem SMPL, protože umožňuje formulovat ztrátovou funkci nad velkým množstvím korespondenčních bodů.

Zároveň má DensePose několik slabin, které se v praxi projeví velmi rychle. Výstup bývá často šumový a místo hladkého povrchu připomíná „lupínky“ (lokálně nekonzistentní přiřazení sousedních pixelů). To vede k tomu, že i při vizuálně správné detekci může být lokální korespondence nekvalitní. Dalším typickým problémem je záměna symetrických částí těla: model může například označit obě nohy jako levé, případně prohodit levou a pravou stranu. V neposlední řadě se chyby objevují i na hranách segmentace (např. u volného oblečení), kde maska zahrne pixely, které ve skutečnosti neodpovídají povrchu těla.

Právě tyto slabiny jsou hlavním důvodem, proč jsem se rozhodl na DensePose navázat. Místo toho, abych DensePose bral jako „konečný“ výsledek, beru jej jako velmi bohatý, ale nedokonalý signál, který je potřeba dále zpracovat a zpřesnit tak, aby byl fitting SMPL stabilní a robustní i v hraničních případech.



Obrázek 2.5: DensePose – Showcase, Použili jsme barevnou mapu (colormap) Jet pro ilustraci. Obě nohy bledě modrou a tedy detekovány jako pravé. Útržek tmavě modré a tedy jiné končetiny je vidět vlevo a odporuje spojitosti těla. Vpravo na obrázku je zase vidět problém s „lupínky“.

3. Praktická část

V této kapitole popisuji praktickou část práce: tři varianty fittingu modelu SMPL do signálu z DensePose. Postup je záměrně explorativní: cílem je rychle ověřovat hypotézy, pojmenovat limity a z výsledků (včetně neúspěchů) vyvodit další směr.

3.1 Společné značení a postup

V textu používám termín *fitting* pro napasování parametrů SMPL tak, aby jeho projekce do obrazu co nejlépe odpovídala výstupu DensePose. Pracuji s DensePose v2 ve variantě Continuous Surface Embeddings, dále jen CSE, která pro každou detekovanou osobu vrací masku S a embedding $\mathbf{e}_p \in \mathbb{R}^D$ pro každý pixel $p \in S$.

Pro SMPL vrcholy existují předpočítané embeddingy $\mathbf{v}_i$. V řadě experimentů proto nejprve převádím DensePose embeddingy na index vrcholu

$$I(p) = \arg\max_i \langle \hat{\mathbf{e}}_p, \hat{\mathbf{v}}_i \rangle,$$

kde $\hat{\mathbf{e}}$ značí L2-normalizovaný embedding. Souřadnici pixelu v obrazové rovině označuji $\mathbf{u}_p \in \mathbb{R}^2$ a projekci SMPL vrcholu i do stejné soustavy jako $\mathbf{x}_i \in \mathbb{R}^2$.

Ve všech metodách optimalizuji stejné proměnné: globální orientaci a posun vůči kameře, pózu θ a tvar β. Jednotlivé varianty se liší pouze ztrátovou funkcí a tím, jakým způsobem z DensePose vytvářejí korespondence pro optimalizaci. Experimenty provádím na snímcích z COCO DensePose (minival) a kvalitu posuzuji kombinací průběhu ztráty a vizuální shody renderovaného SMPL s DensePose výstupem.

3.2 Embedding fitter: loss nad embeddingy a ošetření viditelnosti

3.2.1 Návrh metody

Nejdříve jsem se rozhodl věnovat optimalizační úloze, kde zvolím nějakou chybu pro libovolně promítnutý SMPL model a minimalizací této chyby bych se měl dostat k modelu těla ve 3D, které co nejlépe koresponduje s detekcí člověka v obrázku. Tento nový odhad bych považoval za správnější, jelikož jeho lidská omezení zamezují standardním chybám DensePose.

DensePose CSE mi pro každý pixel postavy nevrátí jen informaci „tady je člověk“, ale také odhad „kde na kanonickém těle tento pixel leží“. Prakticky pracuji s maskou S a embeddingy $\mathbf{e}_p$ pro pixely $p \in S$ (značení viz úvod kapitoly).

Na druhé straně úlohy nějak inicializuji první pozici SMPL. Poté potřebuji vypočítat, jaký „bod na těle“ se v každém pixelu skutečně nachází. To není totéž jako vzít nejbližší promítnutý vrchol, protože projekce do 2D ignoruje zákryty: zadní část těla může být promítnuta do stejné oblasti obrazu jako přední část, ale ve skutečnosti ji kamera nevidí. Proto jsem uvažoval o renderování SMPL

z pohledu kamery, ideálně tak, aby pro každý pixel bylo jasné, který trojúhelník sítě je opravdu viditelný (tj. nejblíže kameře).

Výsledkem renderování pro pixel p je trojúhelník na SMPL síťovině, do kterého paprsek narazil. Trojúhelník popíšu třemi indexy vrcholů (i, j, k). Zároveň dostanu barycentrické váhy $(\lambda_1, \lambda_2, \lambda_3)$ přiřazené vrcholům (i, j, k); platí $\lambda_1 + \lambda_2 + \lambda_3 = 1$. Díky tomu umím z vrcholových hodnot dopočítat hodnotu přímo v bodě průsečíku jako „vážený průměr“ vrcholů.

DensePose CSE má důležitou vlastnost: pro SMPL vrcholy existuje také předpočítaná tabulka embeddingů. Označím ji jako $\mathbf{v}_i \in \mathbb{R}^D$, kde i je index vrcholu SMPL. Pro pixel p pak spočtu embedding bodu na povrchu SMPL, který je v daném pixelu vidět, jako

$$\tilde{\mathbf{e}}_p = \lambda_1 \mathbf{v}_i + \lambda_2 \mathbf{v}_j + \lambda_3 \mathbf{v}_k.$$

Tím mám dvě porovnatelné věci: DensePose embedding $\mathbf{e}_p$ (z obrázku) a „SMPL embedding“ $\tilde{\mathbf{e}}_p$ (z vyrenderované síťoviny). Ztrátovou funkci (loss) pak chci definovat tak, aby byla malá, když oba embeddingy odpovídají stejnému místu na těle. V praxi se embeddingy často normalizují na jednotkovou délku (L2-normalizace), a potom se podobnost měří kosinovou podobností, což je skalární součin:

$$\langle \hat{\mathbf{a}}, \hat{\mathbf{b}} \rangle,$$

kde $\hat{\mathbf{a}}$ je vektor $\mathbf{a}$ po normalizaci (má délku 1). Skvělé na tom je, že hodnota je blízko 1, když vektory míří „stejným směrem“ (tedy jsou si podobné), a menší, když si podobné nejsou. Ztrátu tedy mohu napsat například jako

$$\mathcal{L}_{\text{CSE}} = \sum_{p \in S} \left(1 - \langle \hat{\mathbf{e}}_p, \widehat{\tilde{\mathbf{e}}}_p \rangle\right).$$

Použité značení navazuje na úvod kapitoly; $\tilde{\mathbf{e}}_p$ označuje embedding bodu na povrchu SMPL viditelného v pixelu p. Optimalizace pak probíhá stejně jako u běžného fittingu SMPL: měním parametry SMPL tak, aby se minimalizovala ztráta. Prakticky jde o globální rotaci a posun těla vůči kameře, parametry pózy θ a parametry tvaru β. Cíl je, aby při renderování SMPL vycházelo pro každý pixel „stejné místo na těle“, jaké predikuje DensePose.

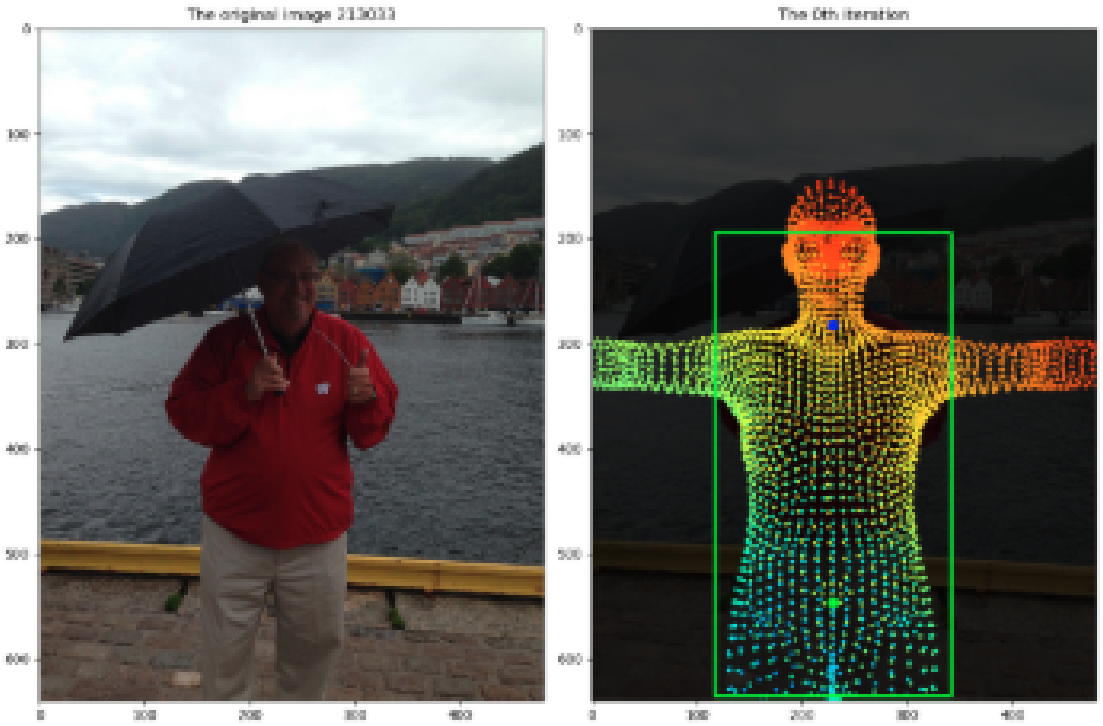
Hlavní očekávaný přínos je v tom, že SMPL funguje jako humanoidní omezení: DensePose může lokálně „šumět“ (nespojitosti, přeskakování mezi částmi těla, záměna levé a pravé strany), ale SMPL vždy musí zůstat jedna konzistentní lidská síťovina s realistickými proporcemi. Fitting tak může DensePose využít jako hustý signál, ale zároveň chyby „vyhladit“ tím, že nejlepší řešení musí být globálně lidské. Vedlejším přínosem je, že výsledkem není jen sada 2D bodů, ale rovnou 3D model těla, ze kterého lze odvodit klouby i prostorovou pózu.

3.2.2 Implementace

V implementaci jsem se snažil co nejpříměji zrealizovat veličiny z návrhu metody: z DensePose beru masku pixelů těla S a embeddingy $\mathbf{e}_p$, ze SMPL v každé iteraci získám projekci do obrazu a pro stejné pixely určuji, který bod na síťovině je z pohledu kamery skutečně vidět. Teprve potom má smysl embeddingy porovnávat, protože bez ošetření zákrytů by do stejné oblasti obrazu „padaly“ i části těla, které kamera ve skutečnosti nevidí.

Nejdříve si připravím DensePose výstup pro jednu detekovanou osobu (bounding box, maska S a embedding pole). SMPL inicializuji pouze heuristicky v neutrální póze a k detekci jej přiblížím

hrubým odhadem posunu pomocí `get_translation` v `common/utils.py`. Tato inicializace je čistě praktická: cílem není odhadnout přesnou pozici, ale dostat model do rozumné hloubky a měřítka, aby optimalizace nezačínala v degenerovaném stavu. Pro to, aby moje metoda začala fungovat mi stačí rozumný překryv těl a dále by se měl program optimalizovat sám.



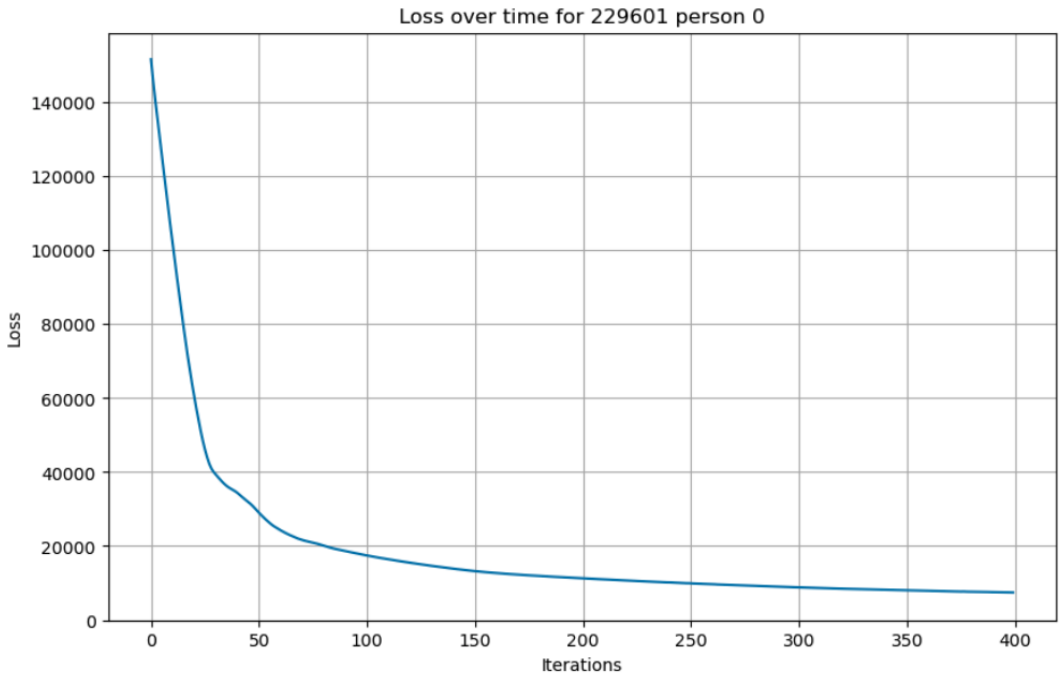
Obrázek 3.1: Příklad inicializace

Pro projekci používám matici intrinsics K sestavenou funkcí `get_camera_intrinsics` (také v common/utils.py). V kódu je použita varianta `K_flipped`, která odpovídá obrazovým souřadnicím (osa y roste směrem dolů), takže se vyhnu časté chybě, kdy se render „zrcadlí“ přes vodorovnou osu. Zároveň jsem musel explicitně ošetřit body za kamerou: čistě matematicky projekce vrátí 2D souřadnice i pro body s $z \leqslant 0$, ale tyto body jsou fyzikálně neviditelné a při optimalizaci působí jako falešné korespondence. Proto v části, která počítá viditelnost, filtruji průsečíky pod minimální hloubkou (v implementaci je na to parametr `min_depth`).

Viditelnost jsem realizoval ray tracingem ve funkci `visible_vertices_gpu`. Prakticky to znamená: pro každý dotazovaný pixel hledám trojúhelník sítě, jehož projekce pixel obsahuje, a z kandidátů vyberu ten s nejmenší hloubkou (nejbližší kameře). Původně jsem chtěl využít barycentrické váhy a embedding interpolovat uvnitř trojúhelníku tak, jak je to v návrhu metody. V prototypu jsem ale nakonec zvolil jednodušší kompromis: z viditelného trojúhelníku vybírám jeden reprezentativní vrchol (nejbližší bodu průsečíku) a embedding beru přímo z jeho předpočítané tabulky. Tím se ztrácí část „plynulosti“ vůči parametrům, ale implementace je výrazně jednodušší a paměťově méně náročná.

Výpočet viditelnosti jsem nejdříve napsal na CPU v NumPy, ale rychle se ukázalo, že je to pro reálné rozlišení příliš pomalé. Přepsání do PyTorch mi umožnilo využít GPU a hlavně vektorizovat celé bloky výpočtu: místo smyček nad pixely se pracuje s tenzory a maskami (typicky tabulka dotazovaných pixelů vůči kandidátním trojúhelníkům), což je v `common/utils.py` vidět i podle značení tvarů v komentářích. Parametry SMPL pak optimalizuji gradientně pomocí Adam, stejně

jako v `projects/euclidean_fitter.py`. Optimalizované proměnné jsou globální orientace, posun, póza θ a tvar β; v každé iteraci přepočtu síťovinu, projekci, viditelnost a následně minimalizuji ztrátu založenou na podobnosti embeddingů na množině pixelů $p \in S$. Počet iterací jsem určil experimentálně na 400. Při větším počtu iterací již nepozoruji žádnou změnu (viz 3.2) hodnoty ztrátové funkce a zbytečně bych prodlužoval čas čekání na výstup.



Obrázek 3.2: Závislost ztráty na iteraci pro daný případ.

3.2.3 Hodnocení

Metoda mi na papíře dávala smysl, protože kombinuje dva silné signály: DensePose poskytuje husté korespondence v obraze a SMPL vynucuje globálně konzistentní lidskou geometrii. V praxi se ale ukázalo, že právě část „pro každý pixel spočti správný bod na povrchu“ je výpočetně i numericky citlivá.

Prvním limitem byla náročnost. Ray tracing nad trojúhelníkovou sítí pro velké množství pixelů se rychle stává úzkým hrdlem a i po přepsání do GPU verze bylo ladění pomalé (zejména pokud jsem chtěl testovat více obrázků nebo vyšší rozlišení masky). Druhý, důležitější problém byla stabilita optimalizace. Opakovaně se mi stávalo, že SMPL vycházel posunutý vůči detekci, případně se optimalizátor „chytil“ špatného lokálního minima a póza se začala hroutit do nelidských tvarů. V této fázi jsem nedokázal jednoznačně izolovat jedinou příčinu, ale prakticky se ukázalo, že metoda je velmi citlivá na inicializaci (hlavně posun a znaménko/škálování hloubky) a na to, jak přesně se ošetří projekce bodů za kamerou. Navíc je zde nepříjemná vlastnost samotné viditelnosti: i malé změny parametrů mohou skokově změnit, který trojúhelník je „první průsečík“, a tím se zhoršuje chování gradientní optimalizace. Z hlediska cíle maturitní práce pro mě bylo klíčové mít metodu, se kterou lze rychle iterovat a která dává interpretovatelné výsledky i na hraničních případech.

Proto jsem tuto variantu vyhodnotil jako slepou uličku a přešel jsem k jednoduššímu měření ztráty v obraze pomocí euklidovské vzdálenosti, které sice obětuje část původní elegance, ale výrazně zlepšuje stabilitu experimentů.

Následující metoda `EuklidovskÃ¡ fitter` vznikla jako zjednodušená varianta předchozího prototypu. Kompletní implementace je v souboru `projects/euclidean_fitter.py`; v `common/utils.py` pak zůstaly pomocné funkce pro práci s projekcí a výpočtem ztráty.

3.3 Euklidovský fitter: loss v obrazové rovině

3.3.1 Návrh metody

V předchozí metodě jsem porovnával DensePose embeddingy se „stejnými" embeddingy na povrchu SMPL, což si ale vyžádalo explicitní řešení viditelnosti (ray tracing) a vedlo to k nestabilní optimalizaci. Zde proto volím jednodušší pohled: DensePose použiji pouze k vytvoření 2D korespondencí mezi pixely a vrcholy SMPL a samotnou ztrátu budu měřit přímo v obrazové rovině euklidovskou vzdáleností.

DensePose CSE mi pro každý pixel postavy dává embedding $\mathbf{e}_p$ a zároveň masku S. Protože pro SMPL vrcholy existují předpočítané embeddingy $\mathbf{v}_i$, mohu pro každý pixel $p \in S$ vybrat nejbližší vrchol (podle kosinové podobnosti, tj. skalárního součinu normalizovaných embeddingů)

$$I(p) = \arg\max_i \langle \hat{\mathbf{e}}_p, \hat{\mathbf{v}}_i \rangle,$$

kde $I(p)$ je index vrcholu SMPL, který DensePose pixelu p přiřazuje. Tím se z DensePose výstupu stane množina 2D bodů s identitou vrcholu: pro každý pixel známe jeho souřadnice $\mathbf{u}_p$ (v souřadnicích bounding boxu) a příslušný index $I(p)$.

Použité značení navazuje na úvod kapitoly.

Na druhé straně mám v každé iteraci optimalizace aktuální SMPL síťovinu. Pro vybranou množinu vrcholů V jejich 3D souřadnice promítnu do obrazu a dostanu 2D body $\mathbf{x}_i \in \mathbb{R}^2$.

Teď nastává praktická otázka: DensePose je pixelová reprezentace, zatímco SMPL vrcholy jsou diskrétní body na síti. Pro jeden vrchol i tak může DensePose poskytnout:

- **žádný pixel** (vrchol se v masce neobjeví, např. protože je zakrytý nebo mimo bounding box),
- **právě jeden pixel**,
- **více pixelů** (typicky proto, že více sousedních pixelů skončí při mapování na stejném vrcholu, případně kvůli šumu DensePose).

Proto si pro každý vrchol i nejprve zavedu množinu pixelů, které mu DensePose přiřadilo:

$$P_i = \{p \in S \mid I(p) = i\}.$$

Vrcholům, pro které platí $|P_i| = 0$, se v ztrátě vyhnu (nemají v obraze žádnou korespondenci). Pokud je pixelů více, volím pro porovnání ten *nejbližší* k projekci vrcholu. Tato volba je záměrně jednoduchá: místo toho, abych penalizoval celý „shluk" pixelů, ptám se, zda se projekce vrcholu dokáže trefit aspoň do některého z pixelů, které k němu DensePose přiřadilo. V praxi to snižuje citlivost na duplicity (více pixelů na jeden vrchol) i na lokální šum.

Ztrátovou funkci pak definuji jako průměrnou vzdálenost mezi projekcí vrcholu a nejbližším DensePose pixelem se stejným indexem vrcholu.

$$\mathcal{L}_{\text{EUC}} = \frac{1}{|V^*|} \sum_{i \in V^*} \min_{p \in P_i} \|\mathbf{x}_i - \mathbf{u}_p\|,$$

kde $V^* = \{i \in V \,|\, |P_i| > 0\}$ je množina vrcholů, které se ve výstupu DensePose vůbec objeví. Norma $\|\cdot\|$ je zde běžná euklidovská vzdálenost v obraze (v pixelech). Celý vzorec lze číst takto: pro každý vrchol, který má v DensePose aspoň jednu 2D korespondenci, vezmu jeho projekci $\mathbf{x}_i$ a porovnám ji s nejbližším pixelem z P_i; tyto vzdálenosti pak zprůměruji.

Metoda se tedy shoduje s předchozím přístupem v tom, že stále využívá DensePose jako hustý signál a SMPL jako globální humanoidní omezení. Liší se ale tím, že se vyhýbá výpočtu viditelnosti na síťovině a porovnávání embeddingů; místo toho optimalizuje přímo jednoduchou geometrickou chybu v obraze. Očekával jsem proto stabilnější chování gradientní optimalizace a výrazně rychlejší iterování experimentů.

3.3.2 Implementace

Implementace v `projects/euclidean_fitter.py` zachovává stejnou kostru jako předchozí metoda: z detekce DensePose vezmu bounding box, masku S a embeddingy, SMPL inicializuji v neutrální póze a optimalizuji proměnné `global_orient`, `transl`, `pose` a `betas` pomocí Adam. Zásadní rozdíl je pouze v tom, jakým způsobem z těchto dat vytvořím ztrátu.

Nejdříve pro každou detekovanou osobu převedu DensePose embeddingy na mapu indexů vrcholů $I(p)$. To zajišťuje funkce `get_closest_vertices_mask_from_ES`, která pro každý pixel v masce vybere vrchol s nejvyšší podobností embeddingu. V kódu tuto mapu počítám ve dvou rozlišeních: v původním rozlišení bounding boxu pro vizualizaci a v menším rozlišení pro samotný výpočet ztráty. Rozlišení pro loss volím tak, aby mělo přibližně konstantní plochu (proměnná `LOSS_AREA`); tím udržuji počet bodů v S a výpočetní náročnost přibližně stejnou napříč různě velkými detekcemi.

Z mapy $I(p)$ si pak připravím dvě pole:

- **`E_indices`** – seznam vrcholových indexů $I(p)$ pro všechny pixely $p \in S$,
- **`E_coordinates`** – odpovídající 2D souřadnice $\mathbf{u}_p$ v souřadnicích zmenšeného bounding boxu.

Volitelně mohu omezit optimalizaci pouze na torso (`TORSO_MASK`), čímž snížím vliv typicky nejhůře predikovaných končetin a zároveň zmenším množinu vrcholů, se kterou loss pracuje.

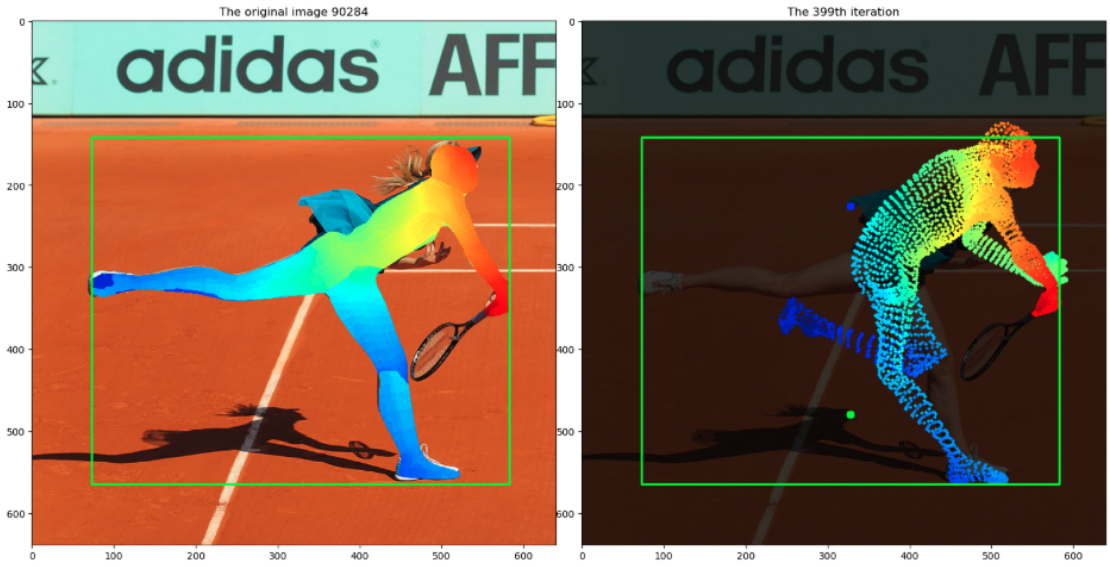
V každé iteraci vyrenderuji (tj. spočtu) aktuální SMPL vrcholy, promítnu je do obrazu pomocí intrinsics matice K z `get_camera_intrinsics` a převedu je do stejné souřadnicové soustavy jako DensePose body: odečtu levý horní roh bounding boxu a aplikuji stejné škálování. Výsledkem je pole `SMPL_coordinates` s body $\mathbf{x}_i$.

Samotnou ztrátu počítá funkce `euclid_loss_gpu` v `common/utils.py`. Naivní CPU varianta by pro každý vrchol i musela prohledat všechny DensePose pixely a najít ty, které mají $I(p) = i$, což vede k pomalým vnořeným smyčkám. GPU verzi jsem proto napsal v PyTorch tak, aby byla plně vektorizovaná: vytvořím tabulku shod indexů (masku tvaru $E \times S$), pomocí broadcastingu spočtu euklidovské vzdálenosti pouze pro shodné dvojice a pro každý vrchol vezmu minimum. Tato část už je analogická postupu z předchozí sekce, kde jsem kvůli výkonu také přecházel od smyček k tenzorovým operacím na GPU.

3.3.3 Hodnocení a výsledky

Z hlediska ladění se tato varianta ukázala jako výrazně příjemnější než předchozí metoda s ray tracingem: ztráta je definovaná čistě v obrazové rovině a výpočet neobsahuje skokové změny viditelnosti trojúhelníků. To ale neznamená, že by metoda automaticky opravovala všechny chyby DensePose.

V případech, kde se DensePose výrazně mýlí (například prohodí končetiny), je pro optimalizaci těžké najít smysluplné řešení, protože korespondence $I(p)$ je už na vstupu nekonzistentní. Typický příklad je na Obrázku 3.3, kde špatně označené části těla vedou SMPL do řešení, které sice lokálně snižuje ztrátu, ale neodpovídá skutečné póze.



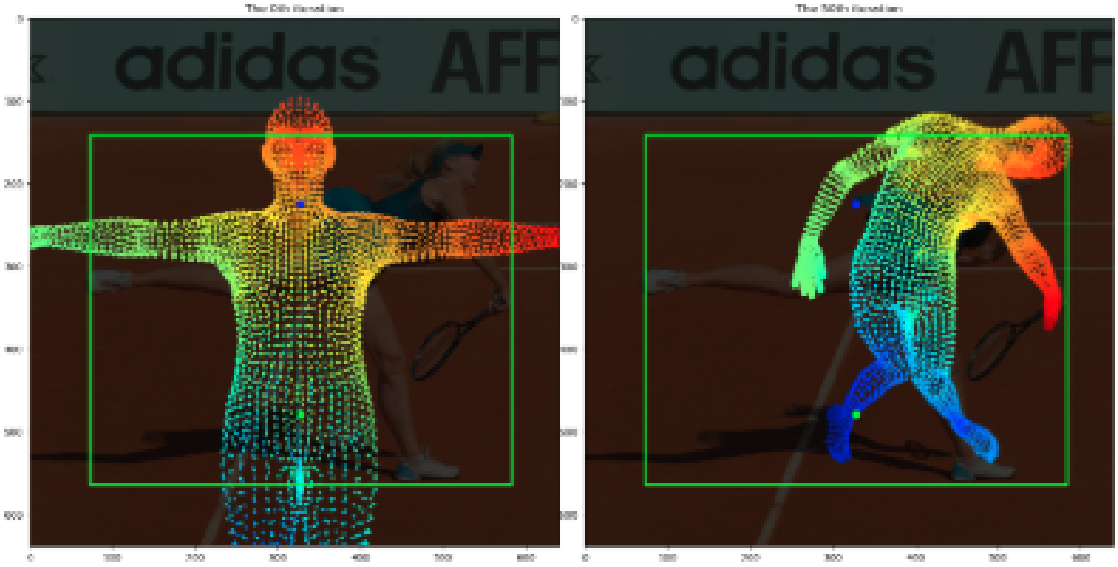
Obrázek 3.3: Výstup DensePose (vlevo) a výstup metody po předposlední (399.) iteraci (vpravo).

Další praktický problém je inicializace. U složitějších scén se optimalizátor často „chytí“ tvarových parametrů β nebo lokálních kloubů dříve, než se podaří správně nastavit globální orientaci a posun. To může vést k deformacím, ze kterých se už optimalizace nevrátí. Typicky se to stává u lidí stojících zády ke kameře (model se musí nejprve celý otočit) nebo u ležících postav. Obrázek 3.4 ukazuje případ, kdy se při použité inicializaci začne tělo deformovat dříve, než se stihne správně zorientovat.

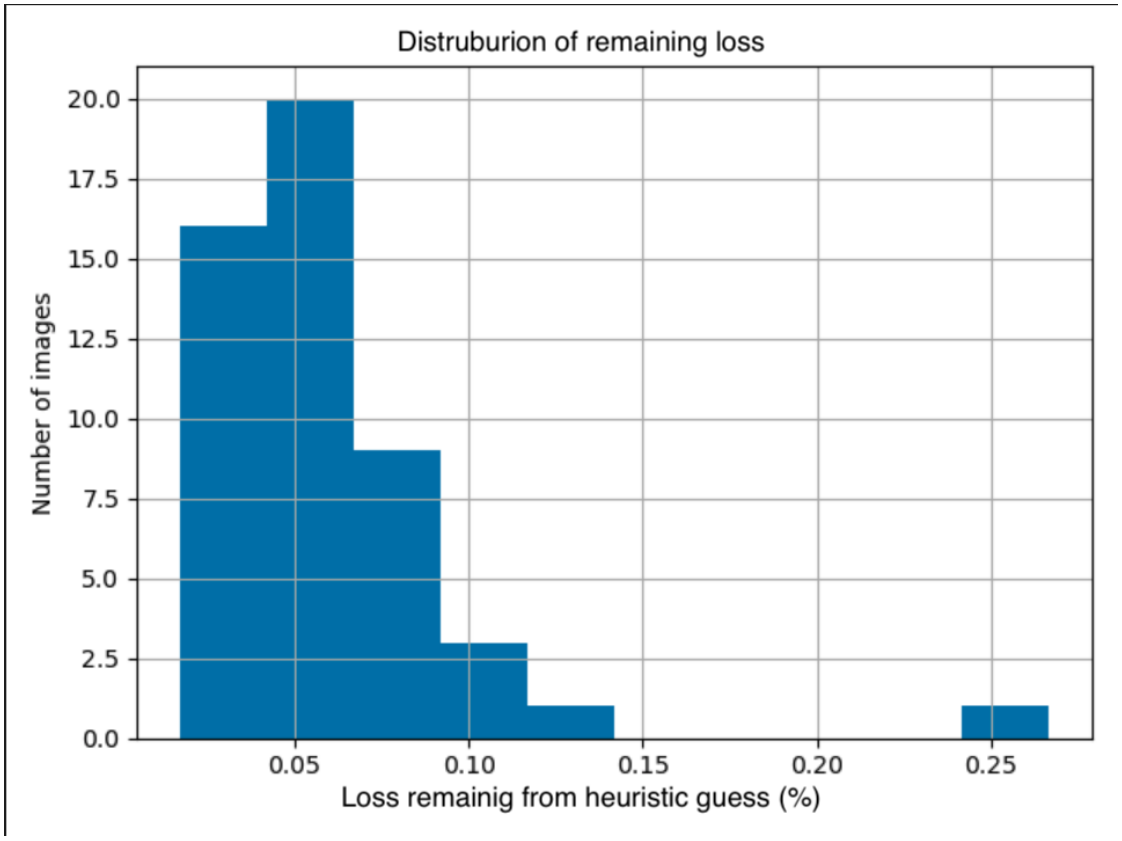
Na druhou stranu se ve velké části případů podaří ztrátu výrazně snížit: pro všechny testované obrázky optimalizace našla konfiguraci SMPL s chybou pouze v jednotkách promile vůči původní heuristické inicializaci (viz Obrázek 3.5). To naznačuje, že zvolená loss funkce je „optimalizovatelná“ a umí SMPL do DensePose dat dotáhnout, pokud se algoritmus nezamotá v prvních iteracích. Zlepšená inicializace (případně silnější regularizace a omezení tvarových parametrů) by proto měla zvýšit počet úspěšných případů.

I za zkoušených podmínek se přibližně u třetiny datasetu podařilo dostat 2D překryv na úroveň původních dat DensePose, s výhodou navíc v podobě explicitního 3D objemu. Zároveň se objevily příklady, kde fitting vede přímo ke zlepšení tvaru a proporcí oproti surovému DensePose výstupu (Obrázek 3.6).

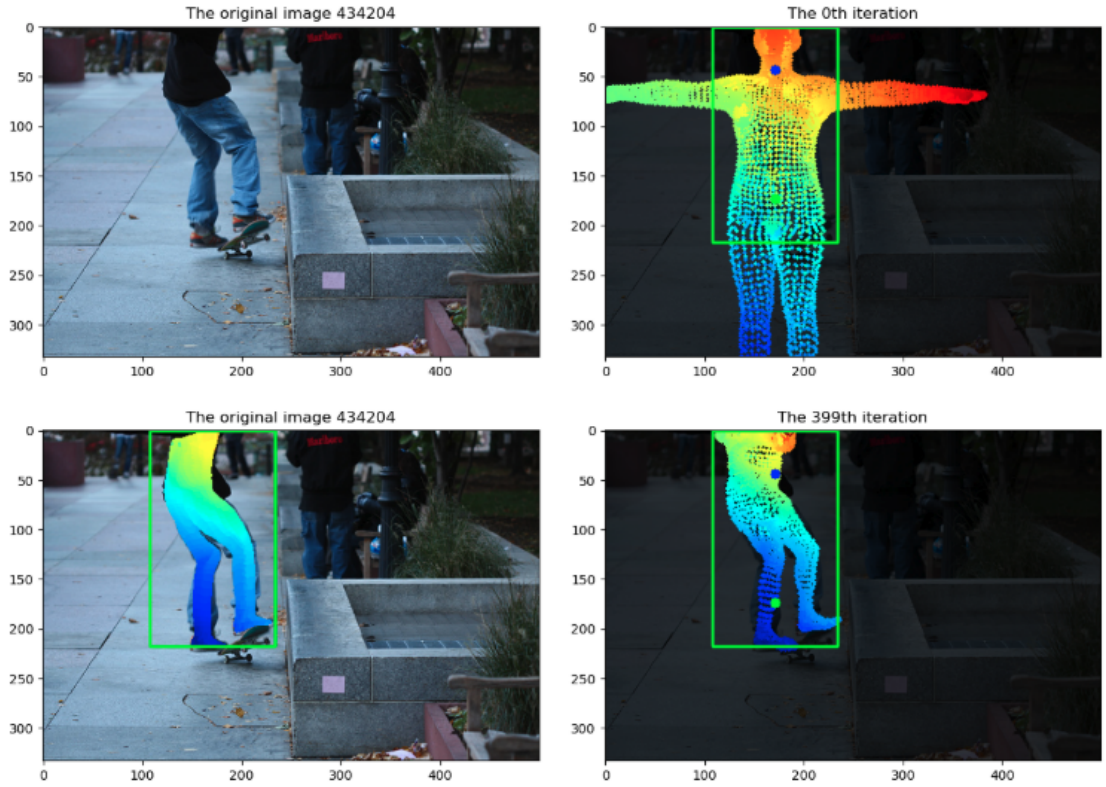
Celkově tato metoda splnila hlavní cíl, který jsem od ní očekával: oproti předchozí variantě je výrazně



Obrázek 3.4: Inicializace (vlevo) a výstup metody po 50. iteraci (vpravo).



Obrázek 3.5: Počet obrázků podle zbývající chyby z původního heuristického odhadu v procentech.



Obrázek 3.6: Příklad vylepšení lidských proporcí

jednodušší, stabilnější na implementaci a umožňuje rychleji provádět experimenty. Její limity jsou ale zřetelné — kvalita výsledku je stále silně závislá na kvalitě DensePose korespondencí a na inicializaci globální transformace. Pro další práci proto dává smysl soustředit se právě na tyto dvě části (lepší inicializace a robustnější práce s chybnými korespondencemi), protože samotná ztráta založená na euklidovské vzdálenosti se v praxi chová předvídatelně a je dobře optimalizovatelná.

Zároveň zde stále platí, že se všemi korespondencemi zacházím stejně: vrchol, který DensePose přiřadí velmi konzistentně, má stejnou „váhu“ jako vrchol, který je v masce rozhozený po více místech. Právě snaha odlišit stabilní korespondence od zjevně problematických mě přivedla k následující metodě `PreciznÃŋfitter`.

3.4 Precizní fitter: vážení korespondencí podle konzistence (prototyp)

3.4.1 Motivace a hypotéza

Precizní fitter je pokus o řešení hlavního nedostatku předchozí metody: v okamžiku, kdy DensePose udělá zásadní chybu v korespondencích (např. označí obě nohy jako levou), dostává optimalizace protichůdné informace a „neví“, kam danou končetinu napasovat. To je dobře vidět i na případech typu Obrázku 3.3, kde špatné korespondence vedou fitting do nelidského lokálního minima.

Základní hypotéza je, že ne všechny DensePose korespondence jsou stejně důvěryhodné, a že část

šumu lze odfiltrovat ještě před samotným fittingem. Zjednodušeně: pokud DensePose přiřazuje jeden SMPL vrchol „na více místech najednou“, je to podezřelé. Proto zavádím jednoduchou míru konzistence korespondence odvozenou z toho, jak moc jsou pixely se stejným vrcholovým indexem v obraze rozptýlené.

3.4.2 Návrh: vážená ztrátová funkce

Nejprve z DensePose embeddingů vytvořím mapu $I(p)$ stejně jako v euklidovském fitteru: pro každý pixel p v masce postavy S vyberu index vrcholu i na SMPL, jehož předpočítaný embedding je k embeddingu pixelu nejbližší.

Značení S, $I(p)$, P_i, $\mathbf{u}_p$ a $\mathbf{x}_i$ navazuje na úvod kapitoly. Nově zavádím střed shluku pixelů $\boldsymbol{\mu}_i$, jeho rozptyl σ_i a z něj odvozenou váhu w_i.

Pro každý vrchol i sesbírám pixely

$$P_i = \{p \in S \mid I(p) = i\}.$$

Pokud DensePose mapuje konzistentně, pak by pixely z P_i měly tvořit malou, souvislou oblast. Naopak při chybách typu prohození končetin, „lupínků“ a šumu na hranách masky se stejný index i začne objevovat na více nesouvisejících místech a P_i bude rozptýlené.

Konzistenci měřím přes střed $\boldsymbol{\mu}_i$ a rozptyl σ_i:

$$\boldsymbol{\mu}_i = \frac{1}{|P_i|} \sum_{p \in P_i} \mathbf{u}_p, \qquad \sigma_i = \sqrt{\frac{1}{|P_i|} \sum_{p \in P_i} \|\mathbf{u}_p - \boldsymbol{\mu}_i\|^2}.$$

Značení $\|\cdot\|$ zde znamená běžnou euklidovskou vzdálenost v pixelech. Prakticky tedy počítám: „vezmi všechny pixely patřící vrcholu i, najdi jejich střed a spočti průměrnou vzdálenost od středu“.

Z rozptylu σ_i odvodím váhu w_i tak, aby platilo: čím větší rozptyl, tím menší váha (menší důvěra). Jednoduchá, hladká volba je

$$w_i = \begin{cases} \exp\left(-\left(\frac{\sigma_i}{\tau}\right)^2\right), & |P_i| \geqslant c_{\min}, \\ 0, & \text{jinak.} \end{cases}$$

Zde τ určuje, jak rychle váha klesá, a $c_{\min}$ potlačí vrcholy, které se v masce objeví jen několikrát (u nich je rozptyl málo vypovídající).

Ztrátu pak počítám v obrazové rovině stejně jako u euklidovského fitteru, ale příspěvky vrcholů vážím:

$$\mathcal{L}_{\text{PREC}} = \frac{1}{\sum_{i \in V^*} w_i} \sum_{i \in V^*} w_i \min_{p \in P_i} \|\mathbf{x}_i - \mathbf{u}_p\|.$$

Zde V^* označuje vrcholy, které se v dané detekci vůbec objevily (tj. mají $|P_i| > 0$). Výraz $\min_{p \in P_i}$ znamená, že pokud DensePose přiřadí vrcholu i více pixelů, beru z nich ten, který je projekci $\mathbf{x}_i$ nejblíže. Dělení součtem vah je normalizace, aby ztráta zůstala ve srovnatelné škále i mezi snímky s různým počtem použitých vrcholů.

3.4.3 Implementovaný prototyp: analýza konzistence korespondencí

Než jsem metodu napojil na samotný fitting, chtěl jsem ověřit dvě praktické otázky: (1) na kterých částech těla jsou DensePose korespondence nejstabilnější (mají nejmenší rozptyl) a (2) zda jsou „přesné body“ rozptýlené po celém těle tak, aby mohly tvořit dostatečnou oporu pro rekonstrukci celé pózy.

Tento krok jsem implementoval jako samostatnou analýzu v souboru `projects/precission_fitter.py`. Skript prochází anotace z COCO DensePose (minival), pro vybrané osoby spustí DensePose CSE a pro každý pixel v masce S určí nejbližší vrchol $I(p)$ pomocí funkce `get_closest_vertices_mask_from_ES`. Následně pro každý vrchol i spočtu $|P_i|$, průměrnou pozici μ_i a rozptyl σ_i.

Výpočet rozptylu je v prototypu plně vektorizovaný v PyTorch a běží na GPU. Využívám `torch.bincount` k tomu, abych pro každý index i získal (1) počet přiřazení, (2) součet x a y souřadnic a (3) součet kvadratických odchylek od průměru, ze kterého vyjde směrodatná odchylka. Výsledky ukládám jako diagnostické vizualizace: (a) překryv DensePose mapování v obraze a (b) 3D vizualizaci SMPL síťoviny, kde jsou vrcholy obarvené podle σ_i. To mi umožňuje rychle zjistit, zda se nízký rozptyl objevuje jen na malé části těla, nebo zda je tato informace použitelná pro fitting jako celek.

V této fázi jsem také uvažoval o normalizaci měřítka (protože σ_i je v pixelech a bez škálování závisí na velikosti bounding boxu). V kódu je tento směr naznačen konstantou `LOSS_AREA`, nicméně plné napojení na zbytek pipeline (volba normalizace, prahování $|P_i|$ a definice vah w_i) jsem už do odevzdání nestihl.

Je důležité zdůraznit, že `projects/precission_fitter.py` proto není „hotový fitter“ v tom smyslu, že by optimalizoval parametry SMPL. Jde o prototyp pro měření a vizualizaci stability DensePose korespondencí, který měl sloužit jako podklad pro finální implementaci vážené ztráty.

3.4.4 Omezení a plán integrace do fittingu

Očekávaný přínos této metody je vyšší robustnost vůči typickým chybám DensePose. Euklidovský fitter pracuje s korespondencemi $I(p)$ jako s fakty; precizní fitter se je naopak snaží doplnit o jednoduchý odhad důvěryhodnosti. Pokud rozptyl opravdu koreluje se správností, pak vážením snížím vliv nekonzistentních částí, které jinak optimalizaci tlačí do špatných řešení.

Současně jde o metodu s jasnými riziky. Nízký rozptyl nemusí znamenat, že je korespondence správná — může jít i o konzistentní, ale systematicky špatné mapování. Rozptyl je nespolehlivý pro vrcholy s malým počtem pixelů ($|P_i|$) a bez normalizace je závislý na velikosti detekce. Hrozí také, že „nejstabilnější“ body budou koncentrované jen na torzu a póza končetin zůstane podurčená; v takovém případě je potřeba kombinovat vážení s dalšími omezeními (regularizace, nebo podmínka pokrytí více částí těla).

Jako přirozené pokračování bych nejprve ověřil samotnou hypotézu kvantitativně: na anotovaných datech porovnat σ_i se skutečnou chybovostí přiřazení vrcholů a určit rozumné prahy a tvar váhové funkce. Teprve potom dává smysl váhy w_i integrovat do fittingu (např. jako váženou verzi euklidovské ztráty) a porovnat stabilitu optimalizace vůči euklidovskému fitteru.

4. Technická dokumentace

Tato kapitola slouží jako praktický návod k instalaci a spuštění repozitáře **SimonPose**, který vznikl jako doprovod k této maturitní práci. Projekt je výzkumný prototyp: není zamýšlen jako obecná knihovna a jeho primárním rozhraním jsou spustitelné skripty v adresáři `projects/`.

Pro běžnou práci jsem si v repozitáři připravil i stručnou uživatelskou dokumentaci v adresáři `docs/` (index `docs/README.md`). Ta je udržovaná spolu se zdrojovým kódem; v této kapitole proto uvádím minimální, samostatně použitelný postup, který stačí k reprodukci experimentů a k pochopení vzniklých výstupů.

4.1 Požadavky na prostředí

Nejdůležitějším předpokladem je funkční GPU stack, protože DensePose i následný fitting SMPL jsou výpočetně náročné. Projekt je otestován primárně na Linuxu s NVIDIA GPU.

- **Python:** 3.11.
- **Operační systém:** Linux doporučen; Windows přes WSL může fungovat; na macOS je běh typicky pouze na CPU a je výrazně pomalý.
- **GPU:** NVIDIA GPU s CUDA (silně doporučené).
- **Disk:** řádově desítky GB (COCO val2014 + modely).
- **Build nástroje:** pro instalaci Detectron2 ze zdrojů je potřeba C++ toolchain a (při CUDA buildu) i CUDA toolkit.

4.2 Struktura repozitáře a očekávané cesty

Repozitář je navržen tak, aby se vše spouštělo z kořene projektu. Cesty k datům a modelům jsou centralizované v souboru `config.py`. Ve výchozím nastavení se předpokládá následující rozložení:

```
data/
  val2014/
    COCO_val2014_*.jpg
  densepose_minival2014_cse.json

models/
  densepose/
    model_final_1d3314.pkl
  smpl/
    models/
      basicmodel_neutral_lbs_10_207_0_v1.1.0.pkl

external/
  detectron2/
```

```
output/
```

Z hlediska čitelnosti kódu je důležité rozdělení na dvě vrstvy:

- **`projects/`** obsahuje spustitelné experimenty (skripty).
- **`common/`** obsahuje sdílené utility (projekce, loss funkce, pomocné DensePose úpravy apod.).

4.3 Instalace

Projekt je nejjednodušší spouštět ve virtuálním prostředí. Postup níže odpovídá minimální instalaci pro `projects/euclidean_fitter.py`.

4.3.1 Lokální instalace

1. **Vytvoření virtuálního prostředí:**

   ```
   python3.11 -m venv .venv
   source .venv/bin/activate
   python -m pip install -U pip setuptools wheel
   ```

2. **Instalace PyTorch:** je potřeba zvolit build odpovídající CUDA runtime. Následující příklad instaluje PyTorch 2.4.0 pro CUDA 12.1:

   ```
   pip install torch==2.4.0+cu121 torchvision==0.19.0+cu121 \
   --index-url https://download.pytorch.org/whl/cu121
   ```

   Pokud si nejsem jistý správnou kombinací, řídím se oficiálním instalátorem: `https://pytorch.org/get-started/locally/`.

3. **Instalace ostatních Python balíčků:**

   ```
   pip install -r requirements.txt
   ```

   Soubor `requirements.txt` záměrně neinstaluje Detectron2 ani DensePose.

4. **Instalace Detectron2 a DensePose ze zdrojů:** repozitář očekává Detectron2 v `external/detectron2/`.

   ```
   mkdir -p external
   cd external
   git clone https://github.com/facebookresearch/detectron2.git

   pip install --no-build-isolation -e external/detectron2
   pip install --no-build-isolation -e external/detectron2/projects/DensePose
   ```

4.3.2 Instalace na clusteru FEL ČVUT (stručně)

Na clusteru je praktické využít modulový systém (PyTorch, OpenCV apod.) a do vlastního `.venv` doinstalovat zbytek. V repozitáři je k tomu připravený návod v `docs/setup-ctu.md`; pro běh je typické i explicitně vybrat GPU přes proměnnou prostředí `CUDA_VISIBLE_DEVICES`.

4.4 Stažení dat a modelových souborů

Z důvodu licencí repozitář neobsahuje datasety ani váhy modelů. Je potřeba je stáhnout a uložit do cest očekávaných v `config.py`. Kompletní, krokový seznam je v `docs/data.md`; zde uvádím klíčové položky:

- **COCO 2014 val** (obrázky): adresář `data/val2014/`.
- **DensePose CSE anotace (minival):** soubor `data/densepose_minival2014_cse.json`.
- **DensePose váhy:** `models/densepose/model_final_1d3314.pkl`.
- **SMPL mapování vrcholů na části těla:** `data/smpl_vert_segmentation.json`.
- **SMPL model:** `models/smpl/models/basicmodel_neutral_lbs_10_207_0_v1.1.0.pkl` (stahuje se ručně po registraci na `https://smpl.is.tue.mpg.de/`).

4.5 Konfigurace (`config.py`)

Soubor `config.py` je záměrně jednoduchý: definuje kořen repozitáře a konkrétní cesty k datům, externím závislostem a modelovým souborům. V praxi jsou pro běh projektů klíčové zejména proměnné `SMPL_MODEL`, `DENSEPOSE_CONFIG` a `DENSEPOSE_WEIGHTS`.

Rychlá kontrola, že se načítají správné cesty:

```
python
from config import *; print(SMPL_MODEL)
print(DENSEPOSE_CONFIG)
print(DENSEPOSE_WEIGHTS)
```

Pokud ukládám data mimo repozitář (typicky na clusteru), mohu buď použít symbolické odkazy na výchozí cesty, nebo `config.py` upravit.

4.6 Spuštění experimentů

Je důležité spouštět skripty **z kořene repozitáře** a **jako modul**. Tím se vyhnu chybám s importy (např. `fromconfigimport*`):

```
python -m projects.euclidean_fitter
```

Každý skript má jednoduché CLI přes `argparse`. Parametry si ověřím pomocí:

```
python -m projects.euclidean_fitter --help
```

4.6.1 `projects/euclidean_fitter.py` (hlavní spustitelný experiment)

Tento skript odpovídá metodě „Euklidovský fitter“ popsané v explorativní části. V jednom běhu: (1) spustí DensePose v2 (CSE), (2) převede embeddingy na indexy vrcholů SMPL a (3) optimalizuje SMPL parametry minimalizací euklidovské chyby v obrazové rovině.

Typické spuštění pro rychlou kontrolu (málo iterací):

```
python -m projects.euclidean_fitter -n 1 -i 10
```

Pro „plnohodnotný“ běh (více iterací):

```
python -m projects.euclidean_fitter -n 1 -i 300
```

Poznámka k implementaci: skript aktuálně zpracuje pouze první detekovanou osobu v obrázku (ve smyčce přes osoby je `break`), což je záměrné zjednodušení pro rychlé ladění.

4.6.2 `projects/precission_fitter.py` (analytický prototyp)

Skript „Precizní fitter“ je v aktuální verzi prototyp pro měření stability DensePose korespondencí (rozptyl pixelů mapovaných na stejný SMPL vrchol). Neprovádí vlastní fitting SMPL; slouží k diagnostice a k návrhu vážené loss funkce popsané v explorativní části.

Spuštění:

```
python -m projects.precission_fitter -n 10
```

4.7 Výstupy

Výsledky všech skriptů se ukládají do adresáře `output/`. Každý běh si vytvoří nový podadresář s automaticky zvolenou příponou (např. `output/euclidean_fitter_0/`), aby se nepřepisovaly předchozí experimenty.

Pro `projects/euclidean_fitter.py` vznikají zejména:

- přehledové obrázky `<image_id>_<person_index>_overview.png` (vývoj ztráty a vizuální diagnostika),

- agregované grafy `loss_histogram.png` a `loss_remaining_histogram.png`.

4.8 Řešení častých problémů

Většina chyb souvisí s citlivým stackem CUDA + PyTorch + Detectron2, případně s chybějícími soubory. Praktické minimum:

- **Chyba „No module named densepose“:** je nainstalovaný Detectron2, ale ne DensePose projekt. Pomůže doinstalovat `external/detectron2/projects/DensePose` (viz instalační kroky).
- **Detectron2 nejde sestavit:** typicky nesedí kombinace PyTorch a CUDA, případně chybí toolchain (gcc/g++, nvcc, ninja). V takovém případě je nejlepší ověřit verze (`torch.version.cuda`, `nvcc--version`, `nvidia-smi`) a sestavit Detectron2 znovu v aktivním `.venv`.
- **Chybí DensePose config nebo váhy:** zkontroluji `config.py` a existenci souborů na disku; typicky jde o proměnné:
  `DENSEPOSE_CONFIG` a `DENSEPOSE_WEIGHTS`.
- **Chybí SMPL model:** SMPL je nutné stáhnout ručně (licence) a případně upravit název souboru v `config.py`.
- **Import chyby při spuštění:** skripty spouštím z kořene repozitáře a jako modul (`python-mprojects...`), ne přímo jako `pythonprojects/...`.

Podrobnější seznam a konkrétní řešení uvádím v `docs/troubleshooting.md`.

4.9 Poznámky k reprodukovatelnosti a licencím

Tento repozitář není benchmarkovací nástroj a cílem je spíše rychlé iterování experimentů. Pokud chci reprodukovat konkrétní obrázky a čísla z této práce, je důležité:

- použít stejné datasety a váhy popsané v `docs/data.md`,
- zaznamenat verze prostředí (PyTorch, CUDA, commit Detectron2),
- držet konzistentní cesty v `config.py`.

Zároveň platí, že COCO, DensePose a SMPL mají vlastní licence a nejsou v repozitáři redistribuovány; při sdílení výsledků je potřeba tyto podmínky respektovat.

5. Závěr

Tato maturitní práce se zabývala možnostmi, jak zlepšit odhad lidské pózy v reálných scénách pomocí kombinace DensePose a parametrického modelu SMPL. DensePose poskytuje hustou korespondenci pixelů na povrch těla, ale jeho výstup je zatížen šumem a typickými chybami (např. záměna levé a pravé strany). SMPL naopak přináší silné humanoidní omezení. Zvolil jsem proto explorativní postup postavený na principu Fail-Fast: rychle navrhnout hypotézu, ověřit ji prototypem a z výsledků (včetně neúspěchů) vyvodit další směr.

5.1 Naplnění cílů

Zadání jsem formuloval realisticky: cílem nebylo vytvořit nový SOTA model, ale metodicky prozkoumat, jak zpřesnit fitting SMPL do signálu z DensePose a jaké překážky se při tom v praxi objevují. Tento cíl se podařilo naplnit ve třech navazujících krocích, které jsou v práci transparentně zdokumentované včetně slepých uliček.

Nejprve jsem navrhl tzv. „Embedding fitter“, který se snaží minimalizovat rozdíl mezi DensePose embeddingem v obraze a embeddingem odvozeným z vyrenderované SMPL síťoviny. Implementace ukázala dvě klíčová omezení: vysokou výpočetní náročnost výpočtu viditelnosti a nestabilitu gradientní optimalizace způsobenou skokovými změnami viditelných trojúhelníků. Výsledek proto nepovažuji za prakticky použitelný v rámci cíle práce, ale jeho analýza byla důležitá pro další návrh.

Na základě těchto zjištění jsem přešel k jednoduššímu a stabilnějšímu přístupu „Euklidovský fitter“. DensePose zde slouží k vytvoření 2D korespondencí mezi pixely masky a vrcholy SMPL; ztráta je pak měřena přímo v obrazové rovině euklidovskou vzdáleností. Tento krok výrazně zlepšil možnost rychle iterovat experimenty a zároveň umožnil v řadě případů získat konzistentní 3D rekonstrukci. Při testování na COCO DensePose (minival) se podařilo přibližně u třetiny zkoušených snímků dosáhnout 2D překryvu srovnatelného s původní DensePose detekcí, přičemž výsledkem je navíc explicitní 3D objem, se kterým lze dále pracovat.

Třetím směrem je „Precizní fitter“, tedy myšlenka pracovat s tím, že ne všechny DensePose korespondence jsou stejně důvěryhodné. V práci jsem implementoval analytický prototyp (`projects/precission_fitter.py`), který měří rozptyl pixelů mapovaných na stejný SMPL vrchol a slouží jako podklad pro váženou ztrátovou funkci. Ačkoli se mi tento krok nepodařilo v časovém rozsahu práce plně integrovat do optimalizace, považuji jej za perspektivní, protože přímo míří na hlavní slabinu euklidovského fitteru: chyby v korespondencích (např. záměna končetin), které mohou optimalizaci stáhnout do nelidského lokálního minima.

Součástí výstupu práce je také repozitář **SimonPose** s implementacemi prototypů a s technickou dokumentací pro instalaci a reprodukci experimentů, publikovaný pod licencí Apache 2.0. Přínos práce tak nespočívá jen ve výsledných obrázcích, ale i v konkrétním, otevřeně sdíleném základu, který umožňuje na metody navázat a systematicky je dále zlepšovat.

5.2 Osobní posun

Během práce jsem si osvojil schopnost číst odborné články a rozumět jejich matematickým formulacím, které pro mě na začátku působily odrazujícím dojmem. Prakticky jsem se naučil pracovat s nástroji a knihovnami, které se v současném počítačovém vidění používají (PyTorch, Detectron2/DensePose, SMPL) a řešit typické problémy spojené s GPU prostředím, výkonem a numerickou stabilitou optimalizace. Zároveň jsem si ověřil, že i neúspěšný prototyp může být užitečný výsledek, pokud je dobře analyzovaný a přetavený do jasných závěrů pro další iteraci.

5.3 Pokračování

Nejpřirozenější pokračování vidím ve dvou praktických směrech: (1) zlepšit inicializaci globální transformace a regularizaci parametrů SMPL tak, aby se fitting nezamotal hned v prvních iteracích, a (2) dokončit vážení korespondencí z „precizního“ prototypu a ověřit, zda opravdu zvyšuje robustnost vůči typickým chybám DensePose. Dává také smysl rozšířit experimenty na více osob v jedné scéně a na video, kde lze využít časovou konzistenci jako další zdroj omezení.

Navzdory technické náročnosti mi práce dávala smysl a postupně mě začala bavit i její psaná část. Projekt proto beru jako otevřený základ, ke kterému bych se rád v budoucnu vrátil a navázal na něj v dalším studiu.

Seznam použité literatury

[MM16] Dmytro Mishkin a Jiří Matas. *All you need is a good init*. 2016. DOI: 10.48550/arXiv.1511.06422. URL: https://arxiv.org/pdf/1511.06422.

[Mor25] Šárka Morávková. *Title of the document*. 2025. URL: https://drive.google.com/file/d/17Mhfoe3TpJCVXy5ry61DvdaYFdgLFuhM/view (cit. 27. 02. 2026).

Seznam obrázků

Seznam tabulek