VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

Fakulta elektrotechniky
a komunikačních technologií

DIPLOMOVÁ PRÁCE

Brno, 2022 Bc. Petr Klimeš

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

ÚSTAV AUTOMATIZACE A MĚŘICÍ TECHNIKY

DEPARTMENT OF CONTROL AND INSTRUMENTATION

VYHODNOCENÍ MRKÁNÍ A POZOROVANÉHO MÍSTA NA MONITORU

EVALUATION OF BLINKING AND THE OBSERVED PLACE ON THE MONITOR

DIPLOMOVÁ PRÁCE

MASTER'S THESIS

AUTOR PRÁCE

AUTHOR

Bc. Petr Klimeš

VEDOUCÍ PRÁCE

SUPERVISOR

Ing. Ilona Janáková, Ph.D.

BRNO 2022

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

Diplomová práce

magisterský navazující studijní program **Kybernetika, automatizace a měření**

Ústav automatizace a měřicí techniky

Student: Bc. Petr Klimeš ***ID:*** 203412

Ročník: 2 ***Akademický rok:*** 2021/22

NÁZEV TÉMATU:

Vyhodnocení mrkání a pozorovaného místa na monitoru

POKYNY PRO VYPRACOVÁNÍ:

Úkolem studenta je navrhnout zařízení pro zpracování obrazu obličeje testované osoby na simulátoru řízení vozidla na UAMT. Cílem diplomové práce je sledování a vyhodnocení frekvence a délky mrkání (pro odhad míry únavy řidiče) a směru pohledu (odhad pozorované oblasti na širokoúhlém monitoru). Předpokládá se realizace v C++, případně Pythonu.

1. Proveďte rešerši v oblasti detekce obličeje, očí, duhovek/zorniček, směru pohledu a mrkání.
2. Navrhněte vhodné zařízení (HW i SW prostředky) umožňující vyhodnocení v reálném čase.
3. Pořiďte dostatečně rozsáhlou a pestrou databázi reálných snímků/sekvencí snímků.
4. Navrhněte možné přístupy zpracování obrazů pro vyhodnocení mrkání a směru pohledu.
5. Zvolené postupy implementujte a otestujte.
6. Řešení implementujte do simulátoru řízení vozidla a ověřte funkčnost měřicího systému.
7. Definujte omezující podmínky. Zhodnoťte.

DOPORUČENÁ LITERATURA:

MAVELY, Annu George, J. E. JUDITH, P. A. SAHAL a Steffy Ann KURUVILLA. Eye gaze tracking based driver monitoring system. In: 2017 IEEE International Conference on Circuits and Systems (ICCS). IEEE, 2017, 2017, s. 364-367. ISBN 978-1-5090-6480-9. Dostupné z: doi:10.1109/ICCS1.2017.8326022

HORÁK, K.; KALOVÁ, I. Eyes Detection and Tracking for Monitoring Driver Vigilance. In The proceedings of the 33rd International Conference on Telecommunication and Signal Processing. H- 1055 Budapest, Szent István krt. 7.: Asszisztencia Szervezo Kft., 2010. p. 204-208. ISBN: 978-963-88981-0- 4.

Termín zadání: 7.2.2022 ***Termín odevzdání:*** 18.5.2022

Vedoucí práce: Ing. Ilona Janáková, Ph.D.

doc. Ing. Petr Fiedler, Ph.D.
předseda rady studijního programu

ABSTRAKT

Cílem práce je vytvořit systém pro analýzu obličeje uživatele simulátoru řízení. Je provedena rešerše existujících přístupů pro detekci významných bodů obličeje, směru pohledu, natočení hlavy a mrkání. Dále je popsán software a hardware simulátoru řízení a je přidána kamera pro snímání uživatele. V praktické části je otestována detekce mrkání pomocí měření Eye Aspect Ratio, detekce natočení hlavy z významných bodů obličeje a odhad směru pohledu pomocí konvolučních neuronových sítí. Z existujících algoritmů je zhodnocena funkčnost knihovny OpenFace. Následně je implementována knihovna OpenFace do Unreal Engine, její funkce jsou využity pro analýzu pohledu i mrkání uživatele. V poslední části je otestována přesnost měření směru pohledu a je porovnáno měření směru pohledu s kalibrací a bez kalibrace.

KLÍČOVÁ SLOVA

Detekce směru pohledu, detekce mrkání, detekce významných bodů obličeje, OpenFace, simulátor řízení, Unreal Engine

ABSTRACT

The purpose of this thesis is to create a system for user's gaze and eye analysis in a driving simulator. There is a research of existing methods for facial landmark detection, head pose and gaze estimation and eye blink detection. Software and hardware of the simulator is described and a camera is added for user recording. In the main part, the Eye Aspect Ratio method is tested for blink detection. Next, head pose detection from facial landmarks is tested. Furthermore, Convolutional Neural Networks are created and trained for gaze estimation. From existing algorithms, OpenFace library is tested. OpenFace library is added to existing software in Unreal Engine and used for gaze and blink detection. A gaze calibration procedure is created and tested for better accuracy.

KEYWORDS

Gaze detection, blink detection, facial landmark detection, OpenFace, driving simulator, Unreal Engine

KLIMEŠ, Petr. *Vyhodnocení mrkání a pozorovaného místa na monitoru*. Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav automatizace a měřicí techniky, 2022, 74 s. Diplomová práce. Vedoucí práce: Ing. Ilona Janáková, Ph.D.

Prohlášení autora o původnosti díla

Jméno a příjmení autora:	Bc. Petr Klimeš
VUT ID autora:	203412
Typ práce:	Diplomová práce
Akademický rok:	2021/22
Téma závěrečné práce:	Vyhodnocení mrkání a pozorovaného místa na monitoru

Prohlašuji, že svou závěrečnou práci jsem vypracoval samostatně pod vedením vedoucí/ho závěrečné práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury na konci práce.

Jako autor uvedené závěrečné práce dále prohlašuji, že v souvislosti s vytvořením této závěrečné práce jsem neporušil autorská práva třetích osob, zejména jsem nezasáhl nedovoleným způsobem do cizích autorských práv osobnostních a/nebo majetkových a jsem si plně vědom následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon), ve znění pozdějších předpisů, včetně možných trestněprávních důsledků vyplývajících z ustanovení části druhé, hlavy VI. díl 4 Trestního zákoníku č. 40/2009 Sb.

Brno ....

podpis autora*

*Autor podepisuje pouze v tištěné verzi.

PODĚKOVÁNÍ

Rád bych poděkoval vedoucímu diplomové práce paní Ing. Iloně Janákové, Ph.D. za odborné vedení, konzultace, trpělivost a podnětné návrhy k práci.

Obsah

Seznam obrázků

Seznam tabulek

Úvod

Simulátor řízení na ústavu automatizace je zařízení, které slouží pro analyzování reakcí uživatele na různé podněty. Příkladem je například reakce na požadavek změny jízdního pruhu, dále reakce na skok zvěře do cesty a podobně. Simulátor se také používá pro zjištění vlivu únavy na řízení řidiče.

Ve výchozím stavu simulátor zaznamenává pouze základní informace jako rychlost vozidla, úhel natočení kol či vzdálenost od požadované trasy. Z těchto informací je však velmi těžké pochopit, proč některý řidič reaguje lépe a jiný hůře. Proto je v této diplomové práci cílem analyzovat snímky z kamery sledující řidiče. Tímto způsobem je možné zjišťovat směr pohledu či frekvenci a délku mrkání a tyto informace dále využít při hodnocení pozornosti a únavy řidiče.

Cílem je tedy vytvořit algoritmus pro detekci směru pohledu uživatele a mrkání na širokoúhlém monitoru s pomocí kamery. Výsledkem je upravený software simulátoru řízení vozidla, který dokáže analyzovat směr pohledu a mrkání uživatele ze snímků.

V první části je uveden výsledek rešerše metod pro detekci obličeje ve snímku, detekce významných bodů obličeje, analýzu směru pohledu a stanovení frekvence mrkání. Dále jsou zkoumány existující implementace algoritmů pro detekci směru pohledu.

Je popsán hardware a software pro zpracování obrazu v simulátoru řízení. Do simulátoru je přidána kamera pro snímání uživatele. Pro testování různých přístupů zpracování obrazu je pořízena databáze reálných snímků.

Dále jsou navrženy a implementovány různé přístupy zpracování obrazu pro vyhodnocení mrkání a směru pohledu.

Knihovna Openface je přidána do software simulátoru řízení. Její funkce jsou využity pro stanovení směru pohledu a mrkání a informace jsou zapisovány do csv souboru. Je otestována kalibrace měření pomocí sledování bodů na obrazovce monitoru. Je zhodnocena funkčnost celého systému.

1 Teoretický úvod

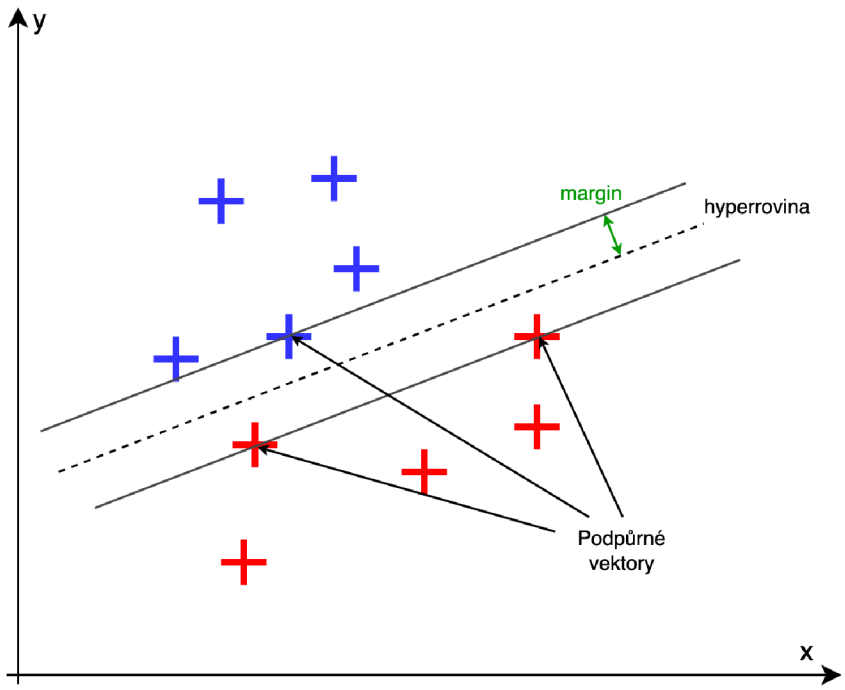
V této části práce bude vysvětleno několik základních metod strojového učení a počítačového vidění využitých v této práci.

1.1 SVM

Support Vector Machine [3] je metoda používaná pro klasifikaci vstupních dat do tříd. Jedná se o metodu učení s učitelem.

Cílem algoritmu je najít hyperrovinu, která optimálně separuje vstupní data. V případě dvourozměrných dat je hyperrovinou přímka, trojrozměrných rovina atd. Podpůrné vektory (support vectors) jsou data, která ovlivňují pozici hyperroviny.

U přímky, separující dvourozměrná data je cílem najít takovou přímku, aby vzdálenost od podpůrných vektorů byla co největší (co největší margin). Princip je zobrazen na obrázku 1.1.



Obr. 1.1: Hyperrovina, separující dvourozměrná data

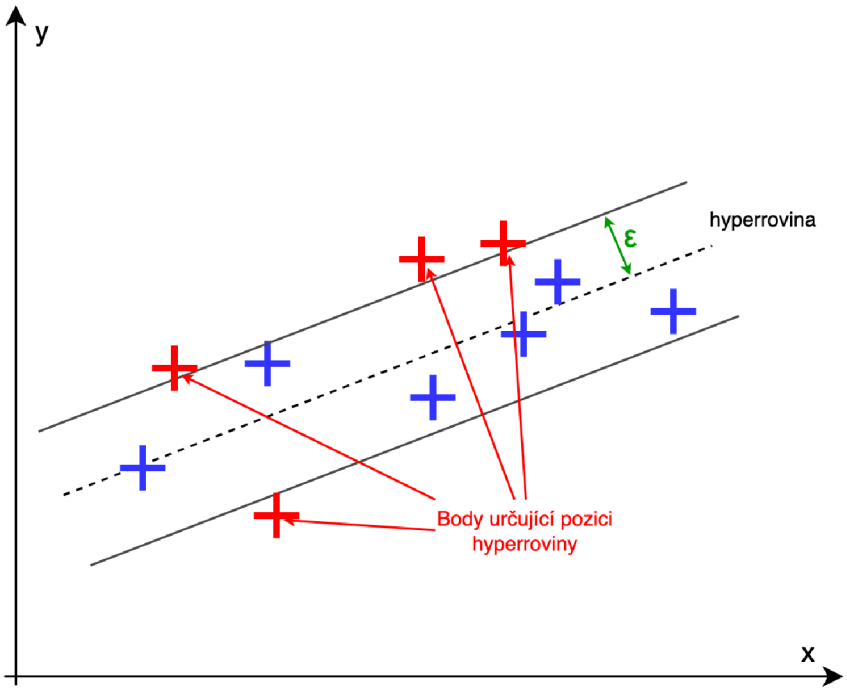
Pokud data nejsou lineárně separovatelná, použije se transformace vstupních dat pomocí jejich součinů do vyšší dimenze, kde už data lineárně separovatelná jsou. Funkce, které zajišťují transformaci součinů do vyšších dimenzí, se nazývají kernely.

1.2 SVR

Support Vector Regression [17] je algoritmus, který vychází z SVM.

Hlavním rozdílem SVM a SVR je, že SVM se používá pro klasifikaci a SVR pro regresi. Parametr ϵ označuje vzdálenost hranice od hyperroviny. Pokud je vzdálenost bodu menší než epsilon, bod není použit pro výpočet hyperroviny. Body vzdálené více než ϵ se označují jako podpůrné vektory a určují pozici hyperroviny (pozice je určena pomocí minimalizace kritéria).

Stejně jako SVM, je možné pomocí kernel funkcí transformovat data do vyšších dimenzí, což je vhodné pro nelineární úlohy.



Obr. 1.2: Support Vector Regression

1.3 Detekce obličeje

[14] V počítačovém vidění se detektorem obličeje rozumí algoritmus, který ve snímku nebo v sekvenci snímků vyhledává obličeje. Cílem algoritmu je většinou vytvořit regiony, které v rámci snímku ohraničují obličeje.

Požadavky na detektor obličeje

- Vysoká úspěšnost klasifikace obličejů
- Malý počet přehlédnutých obličejů

- Malý počet falešně pozitivních
- Co nejmenší výpočetní náročnost

Nejčastěji používané metody

- Haarova kaskáda (VJ detector)
- HOG + linear SVM
- MMOD (max margin object detector)

1.3.1 Viola Jones Detector

Tuto metodu navrhli v roce 2001 Paul Viola a Michael Jones [20]. Často se podle nich nazývá Viola Jones Detector. Jeho hlavní předností je nízká výpočetní náročnost. VJ detektor využívá následující principy:

Haarovy filtry

Jsou to filtry, pomocí kterých se vypočítávají příznaky. Filtr je graficky znázorněn jako černobílý obdélník. Příznak se vypočítá tak, že se suma pixelů v černé části odečte od sumy pixelů v bílé části. Pro obličej je typické například větší jas pixelů v oblasti čela, menší jas očí a úst. Výsledná hodnota příznaku je tak pro danou část obličeje unikátní.

Obr. 1.3: Haarovy prvky kaskády klasifikátorů vybrané metodou AdaBoost [20]

Integrální obraz

Pro rychlý výpočet příznaků se používají integrální obrazy. Integrální obraz vznikne ze snímku podle rovnice 1.1

$$ii(x,y) = \sum_{x' \leq x, y' \leq y} i(x', y') \tag{1.1}$$

S pomocí integrálního obrazu lze velice rychle vyčíslit příznaky z haarových prvků.

AdaBoost

Pro trénování klasifikátoru (vytvoření kaskády haarových prvků) se používá boosting metoda AdaBoost. Funguje tak, že se vytvoří tzv. slabý klasifikátor vstupních dat a zhodnotí úspěšnost klasifikace. Poté neúspěšně klasifikovaná data získají větší váhu a vytvoří se další slabý klasifikátor. Jednoduché klasifikátory jsou pak zkombinovány.

V rámci VJ detektoru skládá AdaBoost kaskádu haar features, které postupně analyzují vstupní obraz. Výhodou je, že pokud vstupem algoritmu není obličej, je okamžitě zamítnut hned prvním klasifikátorem. Pokud na vstupu obličej je, projde všemi haar features a na konci je klasifikován jako obličej.

1.3.2 Detektor obličeje pomocí HOG a SVM

Histogram of Oriented Gradients

[4] Metoda je založena na výpočtu normalizovaných lokálních histogramů orientovaných gradientů. V praxi se změní velikost vstupního obrazu na 128x64 pixelů a pro každé místo se spočítá orientovaný gradient - jeho magnituda a úhel.

Nejprve se vypočítá gradient ve směru x a y:

$$G_x(r,c) = I(r,c+1) - I(r,c-1) \tag{1.2}$$

$$G_y(r,c) = I(r+1,c) - I(r-1,c) \tag{1.3}$$

A z těchto hodnot lze snadno vypočítat orientovaný gradient.

$$Magnitude(\mu) = \sqrt{G_x^2 + G_y^2} \tag{1.4}$$

$$Angle(\phi) = |tan^{-1}(G_y/G_x)| \tag{1.5}$$

Tyto hodnoty se vypočítají pro každý pixel vstupního obrazu. Poté se z bloků obrazu 8x8 vytvoří histogram, který má 9 binů po 20 stupních od 0 do 180. Magnitudy podobně orientovaných gradientů se sčítají a jsou zařazeny do příslušného binu. Dochází k velké redukci velikosti, protože oblast 8x8 lze reprezentovat vektorem o velikosti 9. Histogram je použit pro klasifikaci pomocí SVM 1.1.

1.4 Detekce významných bodů obličeje

Mezi typické významné body obličeje patří špička nosu, okraje očí, tváře, úst apod. Detekce bodů spočívá v určení souřadnic x a y každého bodu.

Úloha najít významné body obličeje může být značně komplikovaná. Mezi výzvy, které se v detekci významných bodů objevují, patří nejčastěji:

- zakrytí části obličeje (například brýlemi, rouškou, jiným objektem)
- různé nasvícení obličeje, barva pleti, makeup
- natočení hlavy - složitá detekce při pohledu z profilu
- výraz - překvapení, smutek, radost, strach - mění konfiguraci významných bodů a zvyšují obtížnost detekce

1.4.1 Datasety

Nejprve je třeba mít kvalitní dataset snímků obličejů, které obsahují anotované souřadnice významných bodů $(x_0, y_0; x_1, y_1; ...x_n, y_n)$ a využít ho pro trénink algoritmů. Mezi nejznámější datasety pro detekci významných bodů patří Helen, 300W(300 Faces in the Wild) či AFW (Annotated Faces in the Wild).

Helen dataset

Helen [13] je dataset vytvořený z platformy Flickr pro sdílení videí a obrázků. Každý snímek obsahuje manuálně vytvořenou anotaci obrysu tváře, nosu, očí, obočí a úst. Dohromady je to 194 bodů pro každý obličej. Helen obsahuje 2000 obrázků pro trénink a 330 pro validaci algoritmu. Příklady snímků z datasetu jsou na obrázku 1.4.

Obr. 1.4: Snímky z Helen datasetu [13]

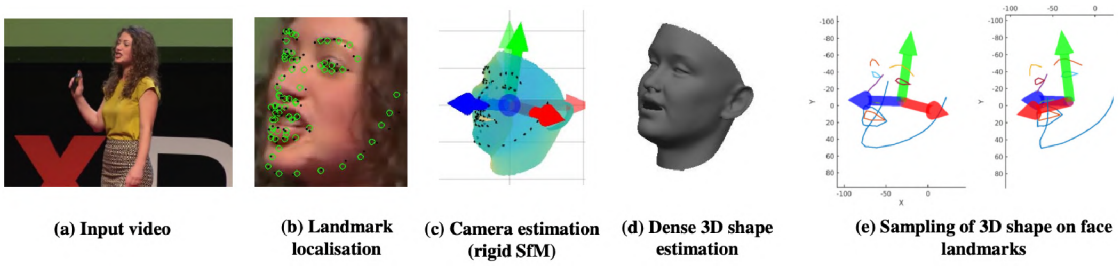
300 Faces in the Wild

300W [16] obsahuje 300 snímků vyfocených uvnitř, 300 vyfocených venku. Pokrývá velké variace výrazů, natočení, nasvětlení, zakrytí části obličeje a velikostí. Snímky jsou staženy z Google. V případě 300W je použita poloautomatická anotace, kdy anotace generuje již naučený model na jiném datasetu a manuálně se pouze kontroluje, že jsou významné body správně umístěny. Výhoda je poté úspora času a stejná konfigurace významných bodů vůči sobě. Celkem je anotováno 68 bodů pro každý snímek.

Menpo

Menpo dataset [26] je jeden z novějších datasetů pro detekci face landmarků (vznikl v roce 2017). Rozdílem oproti ostatním datasetům je, že anotuje obličeje 3D významnými body. Je možné ho využít pro složitější algoritmy, které dokážou určit pozici významných bodů hlavy ve 3D prostoru. Celkem obsahuje 280 000 anotovaných snímků.

Anotace byly vytvořeny automaticky. Proces je vidět na obrázku 1.5. Nejprve jsou pomocí state-of-the-art algoritmu určeny 2D významné body, pomocí kterých je zorientován 3D model obličeje.



Obr. 1.5: Anotace 3D významných bodů [26]

1.4.2 Detektory

Algoritmy, které počítají pozice významných bodů, lze rozdělit do tří kategorií:

1. Holistické metody (Active appearance model)
2. CLM metody
3. Regresní metody

Active appearance model AAM

AAM je statistický model, který se snaží přizpůsobit obrazu obličeje pomocí malého počtu parametrů, které mění vzhled i tvar modelu. AAM obsahuje dvě části - model

tvaru a model vzhledu. Model tvaru se vytvoří tak, že se pro každý anotovaný snímek provede Prokrustova analýza souřadnic významných bodů. Ta zajistí normalizaci bodů (nastaví stejné měřítko, rotaci a posunutí). Poté se použije PCA (Principal component analysis) k redukci dimenzionality. Tímto způsobem se zjistí parametry, které ovlivňují pozici bodů při natočení obličeje, zavření očí, úsměv a podobně. Analogicky se vytvoří i model vzhledu, který pomocí parametrů můžeme měnit. Detekce významných bodů pak spočívá v hledání parametrů vzhledu a tvaru modelu, které co nejlépe odpovídají testovanému snímku obličeje.

CLM

Constrained Local Model počítá pozice významných bodů s pomocí analýzy vzhledu okolí bodu nezávisle pro každý významný bod. Lokální vzhled okolo významného bodu je jednodušší zachytit a také je více robustní vůči změně nasvětlení a zakrytí ve srovnání s AAM.

Stejně jako v případě AAM je v CLM vytvořen model tvaru obličeje, který lze pomocí parametrů nastavovat do všech možných natočení, výrazů, atd. Cílem je najít nejlepší set významných bodů pomocí analýzy pixelů v okolí každého významného bodu a poté upravit body tak, aby vyhovovaly některé konfiguraci modelu tvaru. Algoritmus probíhá následovně:

- Máme regiony $\Omega^{(1)} = \{\Omega_1^{(1)}, \Omega_2^{(1)}, ..., \Omega_D^{(1)}\}$ pro všechny významné body D
- for t = 1:konvergence
    1. V rámci regionu $\Omega_d^{(t)}$ detekuj významný bod zvlášť s pomocí lokální analýzy vzhledu regionu. Výsledky detekce zapiš jako souřadnice m^t.
    2. Uprav pozice m tak, aby vyhovovaly konfiguraci modelu tvaru a ulož odhadované souřadnice významných bodů x^t či parametry modelu tvaru p^t
    3. Uprav každý region $\Omega_d^{(t+1)}$ tak, aby byl v okolí aktualizovaných významných bodů

(a) Start (b) After It1 (c) After It2 (d) Final Pts

Obr. 1.6: Detekce pomocí CLM[1]

Regresní metody

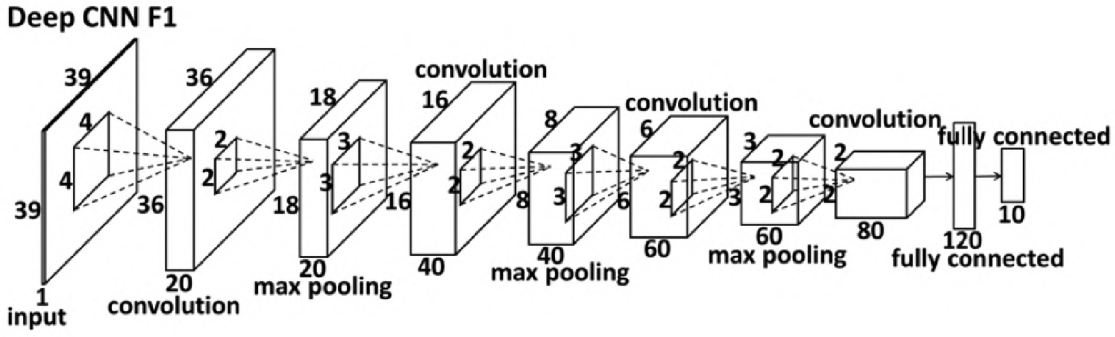
Oproti předchozím přístupům regresní metody přímo mapují významné body ze vstupního snímku. Metody lze rozdělit na:

1. Přímé
2. Kaskádní
3. Založené na hlubokém učení

Přímá regrese probíhá ze vstupního snímku bez inicializace souřadnic významných bodů v jedné iteraci. Regresor se naučí odhadovat významné body z obrazu.

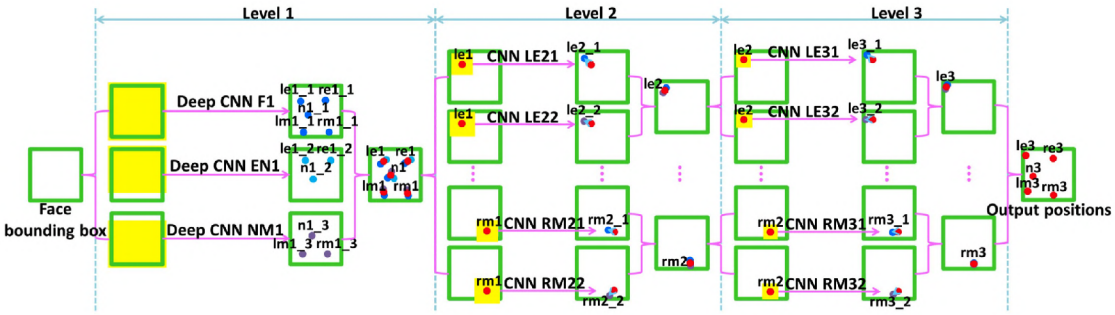
Mezi kaskádní regresní metody patří regresní stromy. Tato metoda je použita pro detektor významných bodů knihovny dlib. Pro detekci se používá kaskáda regresorů, což jsou funkce, které na základě snímku obličeje a tvaru (aktuální pozici) významných bodů průběžně významné body aktualizují, dokud není splněno kritérium.

Metody založené na hlubokém učení jsou v poslední době velice populární. Pro detekci významných bodů se používají konvoluční neuronové sítě, jejichž vstupem je snímek obličeje a výstupem množina n významných bodů $\{x_1, y_1, ..., x_n, y_n\}$. Architektura neuronové sítě obsahuje konvoluční filtry, kterými se postupně extrahuje informace o pozici bodu. Příkladem architektury je neuronová síť na obrázku 1.7. Tato síť o 10 výstupech dokáže odhadovat pozici pěti významných bodů. V práci [18] jsou dále v kaskádě použity neuronové sítě pro každý bod, které doladí pozici významného bodu a zlepšují kvalitu predikce. Architektura kaskády CNN je na obrázku 1.7



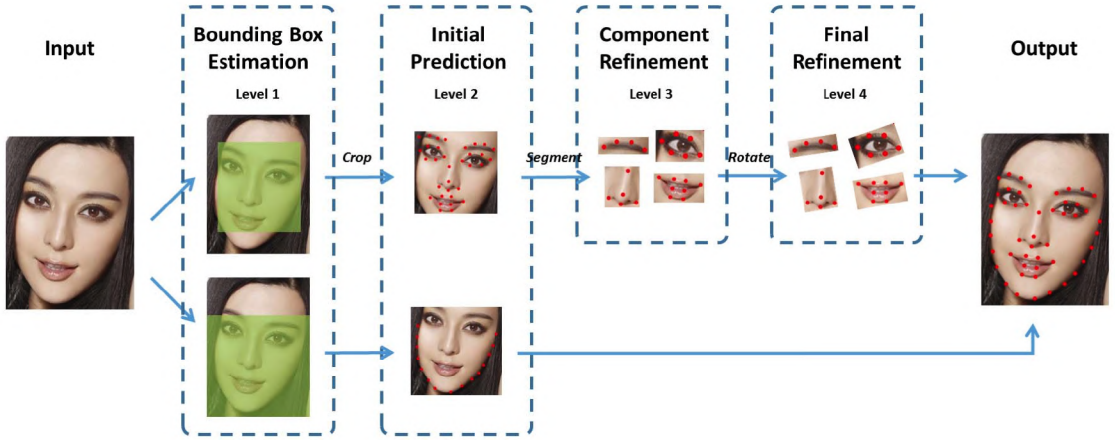
Obr. 1.7: Příklad architektury neuronové sítě pro detekci významných bodů [18]

Přístup kaskádního spojení několika neuronových sítí je i v další práci [29], kde první úroveň predikuje oblast snímku, ohraničující kontury obličeje a oblast ohraničující pouze vnitřní body. V druhé úrovni jsou vytvořeny predikce, které se v případě vnitřních bodů ještě zpřesňují v dalších úrovních, kde se s pomocí prvotní predikce



Obr. 1.8: Kaskádní architerktura algoritmu pro detekci významných bodů [18]

významných bodů vyříznou části snímku reprezentující část obličeje a ty se rotují do normalizované konfigurace. Princip je zobrazen na obrázku 1.9.



Obr. 1.9: Princip určení významných bodů [29]

1.4.3 Využití landmark detektorů [9]

Detekce významných bodů může být použita pro animaci obličeje postav v animovaných filmech či úpravu pohybu úst při dabování do cizího jazyka.

Významné body obličeje mohou pomoci i v asistenčních systémech řidiče, kdy z jejich pozice dokážeme určit únavu. Pro tuto úlohu můžeme použít neuronovou síť, jejíž vstupem bude pozice landmarků a výstupem únava.

1.5 Detekce očí

[10] Při analýze směru pohledu je nutné nejprve detekovat oči a z nich extrahovat informaci o směru pohledu. Není to jednoduchá úloha, protože vzhled očí je významně

ovlivněn směrem pohledu. Další vlivy, které ovlivňují vzhled oka, jsou barva pleti člověka, barva duhovky, světelné podmínky a stav otevření oka (zda je otevřené, přivřené či zavřené). Detekci očí většinou zajišťují detektory významných bodů, popsané v předchozí kapitole. Všechny landmark detektory určují body ohraničující oči, a proto není třeba vytvářet speciální modely.

1.6 Detekce směru pohledu

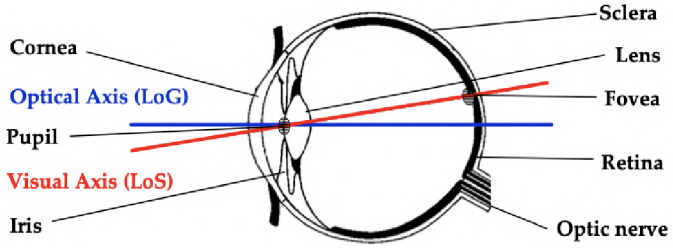
[10] [15] Detekce směru pohledu je úloha, která v posledních letech nabývá na popularitě. Algoritmy pro detekci směru pohledu nacházejí využití v automobilovém průmyslu, kde se využívají pro analýzu pozornosti řidiče. Například vozidla Subaru mají systém s názvem Driver Monitoring System, který pomocí dvou kamer umístěných nad infotainmentem sleduje řidiče a upozorňuje ho ve chvíli, kdy se přestane soustředit na řízení. Tímto se zvyšuje bezpečnost automobilu.

Další využití může být pro sledování zájmu na webových stránkách. Algoritmus určí místo pohledu na obrazovce (POG - Point Of Gaze) a tato informace může být využita pro vylepšení grafického rozložení stránky tak, aby uživatel sledoval to, co tvůrce potřebuje.

Dále je směr pohledu možné využít pro ovládání počítačů pohledem, to zahrnuje zadávání textu nebo pohyb myší. Smartphone nebo tablet může obsahovat sledování pohledu pro zhasínání displeje, pokud se uživatel nedívá.

1.6.1 Struktura oka a pohyby

Struktura lidského oka používaná v aplikacích pro sledování pohledu je na obrázku 1.10. Algoritmy modelující směr pohledu jsou založeny buď na modelování optické osy (Optical Axis) nebo osy vidění (Visual Axis). Optická osa prochází středem zornice, rohovky a oční bulvy. Střed rohovky je nazýván uzlový bod oka. Optická osa a osa vidění se protínají v tomto bodě pod určitým úhlem. Směr pohledu je definován osou vidění.



Obr. 1.10: Struktura lidského oka [10]

Mezi základní pohyby očí při sledování obrazovky patří sakády a fixace. Sakády jsou charakterizovány jako přesuny pohledu mezi jednotlivými fixacemi. Probíhají ve chvíli, kdy člověk na monitoru něco hledá. Rychlost sakád je 100-700° za sekundu.

Další kategorií pohybů očí jsou fixace. Fixace jsou fáze, kdy se člověk soustředí na nějakou obrazovou informaci.

1.6.2 Metody pro detekci směru pohledu

[10] Algoritmy, které detekují směr pohledu, většinou určují horizontální a vertikální úhel pohledu, kde [0, 0] znamená, že se uživatel dívá přímo do kamery. Pro odhad směru pohledu se používá detekce natočení hlavy (tzv. head pose) z významných bodů obličeje a detekce směru pohledu z detailu snímku oka.

Detekce směru pohledu na obrazovku spočívá v určení místa, které zkoumaná osoba sleduje. Místo se v literatuře označuje jako POG (Point Of Gaze - bod pohledu). Pokud je cílem detekovat POG, samotné úhly pohledu nestačí, je nutné také znát pozici hlavy v 3D prostoru, abychom mohli detekovat průsečík vektoru pohledu s monitorem.

Úloha detekce směru pohledu má několik výzev. Mezi ně patří:

1. Zakrytí části očí očními víčky, mrkání
2. Různá barva duhovky (více či méně kontrastní)
3. Kontaktní čočky, dioptrické brýle

Rozdělení metod pro detekci směru pohledu:

- Podle umístění kamery:
    1. Blízko oka (většinou ve formě brýlí vybavených kamerami, které snímají okolní scénu a detail oka)
    2. Pomocí pevně umístěné kamery (typický příklad je detekce pohledu na monitor, vůči kterému je kamera pevně umístěna)
- Podle metody zpracování obrazu:
    1. Feature based
    2. Model based
    3. Appearance based
    4. Cross-Ratio based

V kontextu úlohy je zřejmé, že bude výhodnější kameru pevně umístit nad monitor, protože bude snazší ji zapojit do systému počítače a uživatel nebude muset mít nasazené brýle. Nevýhodou může být menší kvalita obrazu oka a nutnost implementace algoritmu rozpoznávající směr vidění do software simulátoru.

Nevýhodou detekce pomocí jedné kamery umístěné pevně nad monitorem bývá složitá přesná detekce směru pohledu při větším natočení hlavy. Některé přístupy

proto s pomocí jedné kamery odhadují směr pohledu pouze v omezeném rozsahu.

Feature based metody

Mezi feature-based metody patří extrakce příznaků souvisejících se směrem pohledu uživatele. Tyto obrazové informace (souřadnice významných bodů obličeje, střed zornice, okraje očí) se použijí přímo k regresi pomocí polynomu, SVR nebo neronové sítě. Výstupem je poté nejčastěji směr pohledu ve formě dvou úhlů.

Model based metody

Metody založené na modelu používají geometrický popis modelu oka k výpočtu směru pohledu. Bod pohledu je určen jako průsečík směru pohledu se sledovaným objektem. Cílem je ze snímku určit střed oční bulvy, zornice a na základě toho vytvořit vektor, který popisuje směr pohledu. Protože osu vidění nelze z obrazu oka určit, úhel mezi optickou osou a osou vidění je určen kalibrací.

Appearance based metody

Jsou to metody, které jsou založené na vzhledu oka. Pro určení směru pohledu tímto způsobem se často používají konvoluční neuronové sítě. Základem těchto metod jsou velké datasety anotovaných snímků, na kterých se CNN trénují. Mezi nejznámější patří:

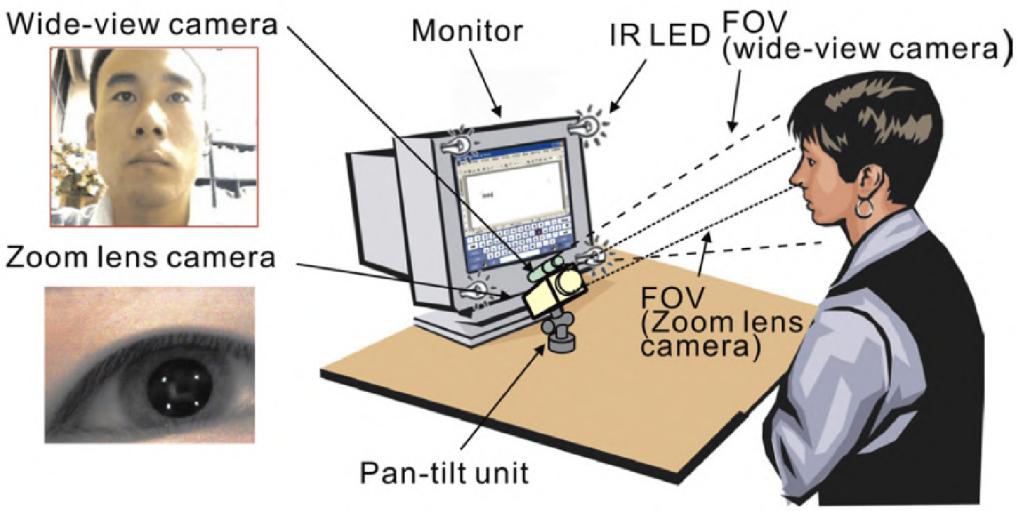
- MPIIGaze
- GazeCapture
- SynthesEyes
- UnityEyes
- NVGaze

Cross-Ratio based metody

Využívají nasvětlení pomocí neviditelného IR světla. Ve snímku se pak detekují jejich odrazy na rohovce. Tyto metody jsou robustní, avšak vyžadují další hardware v podobě IR zdrojů, například v rozích monitoru.

Princip, který byl použit v [25], je vidět na obrázku 1.12. V rozích monitoru jsou 4 IR diody, které vysílají paprsek světla na rohovku, další IR zdroj je v ose kamery pro vytvoření tzv. Bright Pupil efektu. Jsou použity dvě kamery, první se širokoúhlým objektivem zabírá celý obličej, druhá se zoom objektivem snímá detail oka ve vysokém rozlišení. IR zdroje v rozích monitoru vytvářejí odraz na rohovce a pomocí Bright Pupil je určen střed zornice. Získáme 5 bodů promítnutých do

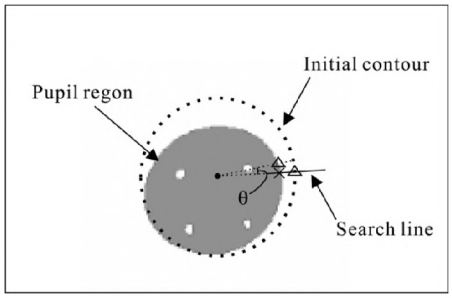
obrazové roviny kamery, kde 4 odpovídají rohům monitoru a jeden určuje střed zornice (směr pohledu)



Obr. 1.11: Cross Ratio princip [25]

Jsou aproximovány virtuální projekce bodů do roviny, která je tečná ke středu zornice. Tyto body lze pomocí cross ratio metody přepočítat na body na obrazovce. Metoda je velmi závislá na přesném určení středu zornice. Pro tento účel byl použit následující algoritmus:

1. rozdílový snímek bright pupil a dark pupil (vytvořen vypnutím IR světla kamery)
2. prahování rozdílového snímku, pokud je intenzita větší jak práh 1, jinak 0
3. výpočet středu a poloměru zornice nahrubo - střed je centroid oblasti a poloměr je určen tak, aby měla oblast stejný počet pixelů
4. detekce okraje zornice na normálách kružnice pomocí vektoru [0 0 0 1 1 1]
5. proložení detekovaných bodů elipsou a nalezení středu elipsy

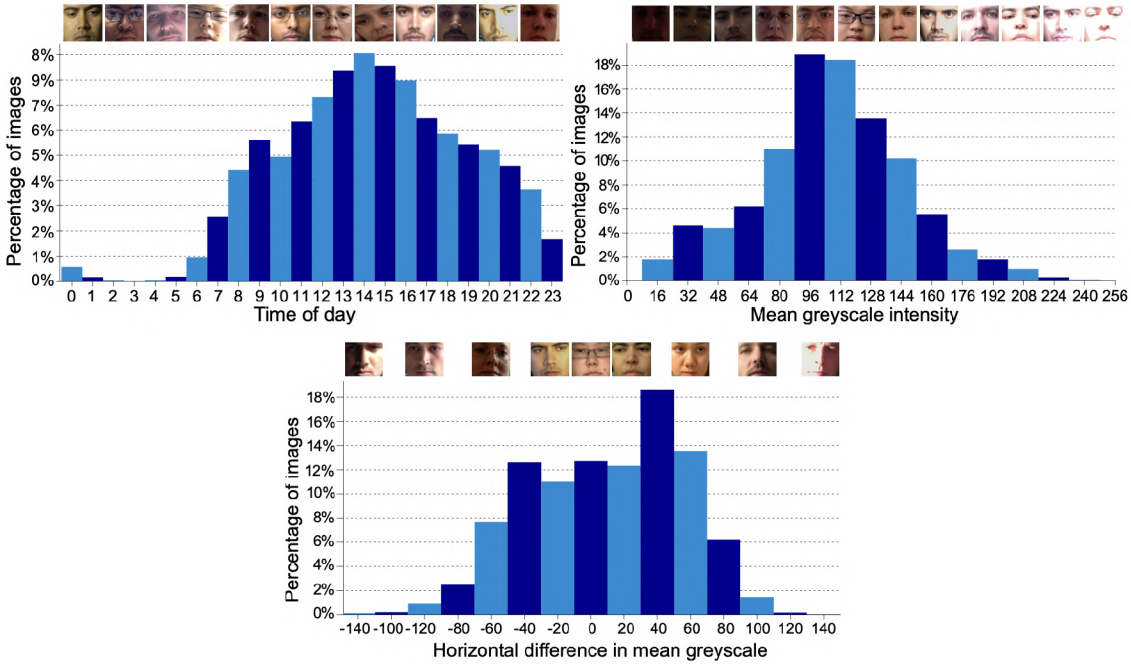


Obr. 1.12: Hledání středu zornice [25]

1.6.3 Datasety

MPIIGaze

Tento dataset je tzv. in the wild, to znamená, že podmínky (osvětlení, kamera) nejsou nijak omezeny. Na obrázku 1.13 je vidět, že snímky jsou variabilní. Jsou zde jak velmi tmavé snímky i tzv. přepálené snímky, na třetím grafu je vidět i různá nasvětlení ze strany.



Obr. 1.13: Variabilita MPIIGaze datasetu [28]

Proces tvorby datasetu probíhal s pomocí notebooků. Uživatel nahrával snímky s pomocí software, který ho jednou za každých 10 minut požádal, aby nahrál snímky. Uživateli se zobrazovaly šedé body na 20 místech na obrazovce. Každý bod se vždy zmenšoval, až úplně zmizel a uživatel měl za úkol vždy zmáčknout mezerník přesně ve chvíli, kdy měl bod zmizet. Tímto způsobem se kontrolovalo, že uživatel bod sleduje.

Dataset celkem obsahuje 213659 snímků od 15 osob.

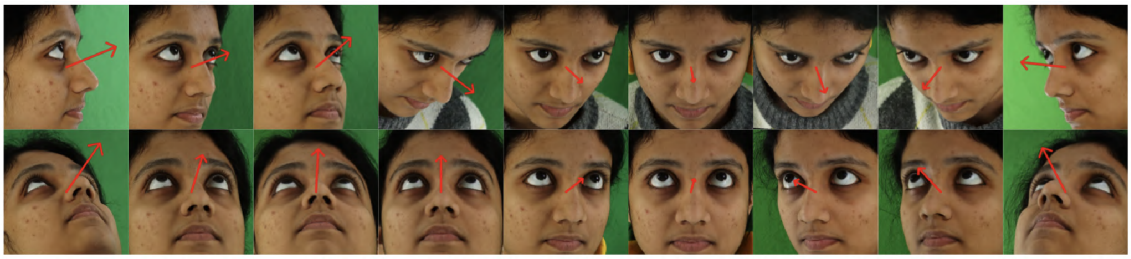
EYEDIAP

Dataset [5] je specifický tím, že byl nahrán pomocí RGB i RGBD kamery, takže záznamy obsahují i informaci o vzdálenosti od kamery. Další specifikum je v tom, že je v datasetu rozlišeno sledování pouze očima a sledování včetně natočení hlavy. Část snímků byla tvořena tak, že uživatel sledoval náhodné body na obrazovce 24

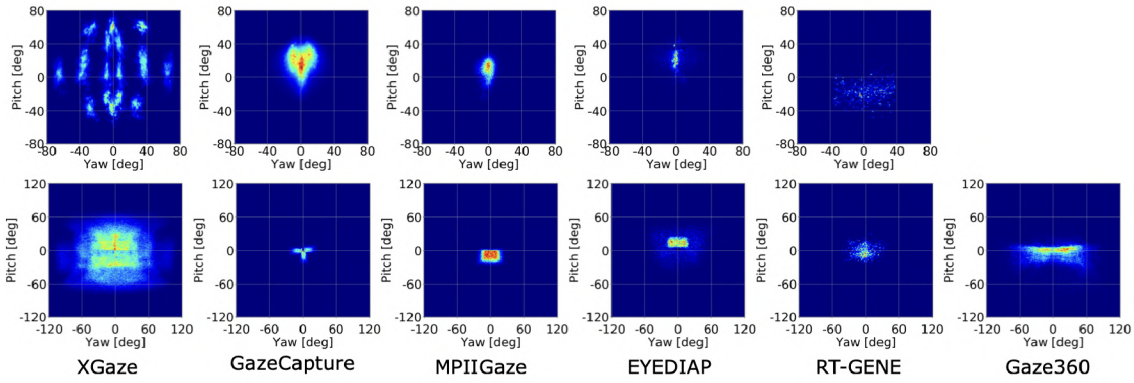
palcového monitoru, na části snímků se uživatel díval na 3D objekt (koule o průměru 4 cm) zavěšený na silonu.

ETH-XGaze

XGaze [27] je dataset, který se od ostatních liší hlavně velkým rozsahem směru pohledu uživatelů ve snímcích. Dataset je vytvořen s pomocí 18 zrcadlovek Canon EOS 250D, které jsou rozmístěny tak, že snímají uživatele z 18 různých úhlů. Tímto způsobem se mimo jiné urychluje sběr snímků, protože při každém natočení hlavy uživatele vznikne 18 snímků o různém směru pohledu. Uživatel sleduje bod promítaný na plátno. Proces sledování bodu je zde velice podobný jako u MPIIGaze. Nasvětlení scény je tvořeno pomocí lightboxů. Celkem má dataset 1,083,492 snímků od 110 uživatelů. Jeho další výhodou je vysoká kvalita obrazu v rozlišení 6000x4000 px.



Obr. 1.14: Příklad směru pohledu zachyceného z 18 kamer [27]



Obr. 1.15: Srovnání rozložení směru pohledu v rámci datasetů [27]

Syntetické datasety

Snímky syntetických datasetů nejsou vytvořeny klasickou cestou snímkování kamerou, ale s pomocí 3D modelovacích nástrojů. Základem syntetických datasetů pro

sledování směru pohledu je 3D model oka, který je nejčastěji vytvořen pomocí retopologie 3D skenu. Výhodou je, že můžeme využít automatickou anotaci, protože anotace odpovídá parametrům modelu. Nevýhodou může být horší predikce natrénovaných modelů kvůli tomu, že snímky vznikají pomocí počítačového algoritmu, a tak nemusí přesně odpovídat reálnému vzhledu.

Mezi nejznámější syntetické datasety pro určení směru pohledu patří SynthesEyes [22] a UnityEyes [23], dále také NVGaze od společnosti NVIDIA.

1.6.4 Komerčně dostupné eyetracker moduly

Tobii Eye Tracker 5

Eye tracker 5 je zařízení, které lze namontovat podobně jako webkameru na monitor. Obsahuje mikro projektory, které promítají NIR světlo a detekují jeho odraz na rohovce pomocí kamery. Dále obsahuje integrovaný chip, který zajišťuje zpracování obrazu potřebné pro detekci směru pohledu. Pomocí USB kabelu se tento modul připojí k počítači a nejčastěji se využívá pro natáčení FOV(field of view) ve hře. Tímto dokáže simulovat pohled ve VR brýlích. Směr pohledu dokáže detekovat v rozmezí 40° horizontálně i vertikálně s frekvencí 133Hz. Tobii Eye Tracker je možné zakoupit za 6990 Kč.

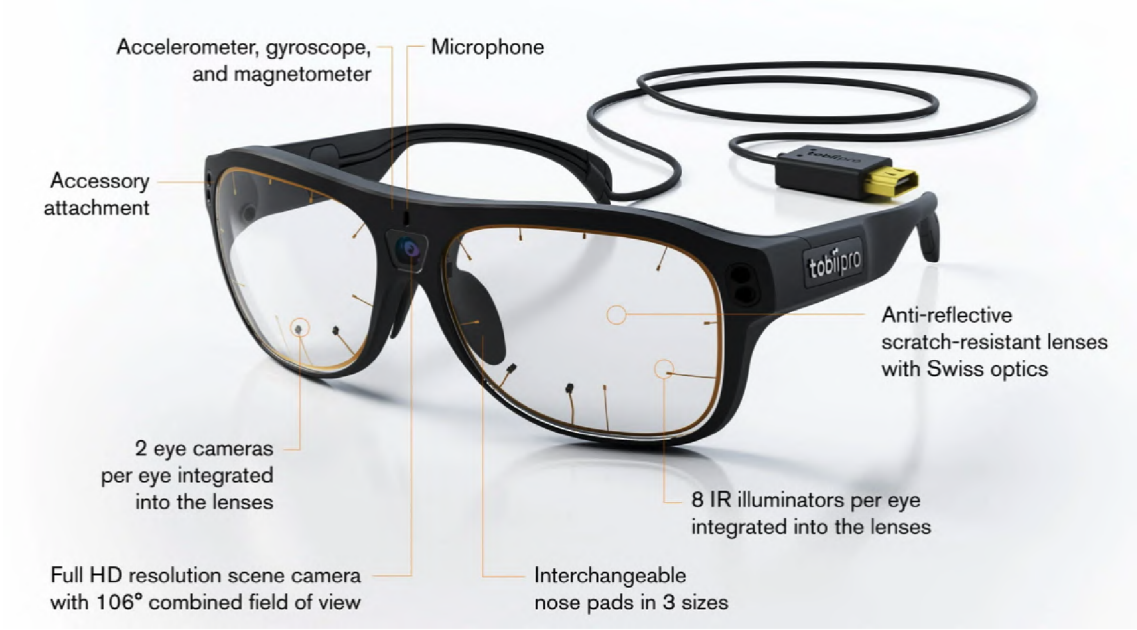
Obr. 1.16: Tobii Eye Tracker 5 `https://tech.tobii.com`

Tobii Pro Glasses 3

Další produkt od Tobii je tzv. wearable eye tracker ve formě brýlí, které sledují oči uživatele pomocí čtyř kamer a okolní scénu pomocí kamery uprostřed. Používají stejný princip s nasvícením NIR světlem a detekcí odrazů na rohovce. Brýle jsou poměrně nenápadné, na první pohled připomínají klasické brýle. Využití nacházejí při různých výzkumech chování sportovců `https://www.youtube.com/watch?v=JNqE87_7bOg`.

Smart Eye AI-X

AI-X je eye tracker společnosti IMotions, který dokáže sledovat pohled uživatele s přesností 0.5°. Je to podobné zařízení jako Tobii Eye Tracker. Také využívá odrazů NIR světla od rohovky pro určení směru pohledu. Omezením je velikost monitoru, který může být maximálně 24 palců. V ČR ho nelze zakoupit.



Obr. 1.17: Tobii Pro Glasses 3 `https://www.tobiipro.com/`

Obr. 1.18: SmartEye AI-X `https://smarteye.se/ai-x/`

1.6.5 Kalibrace měření směru pohledu

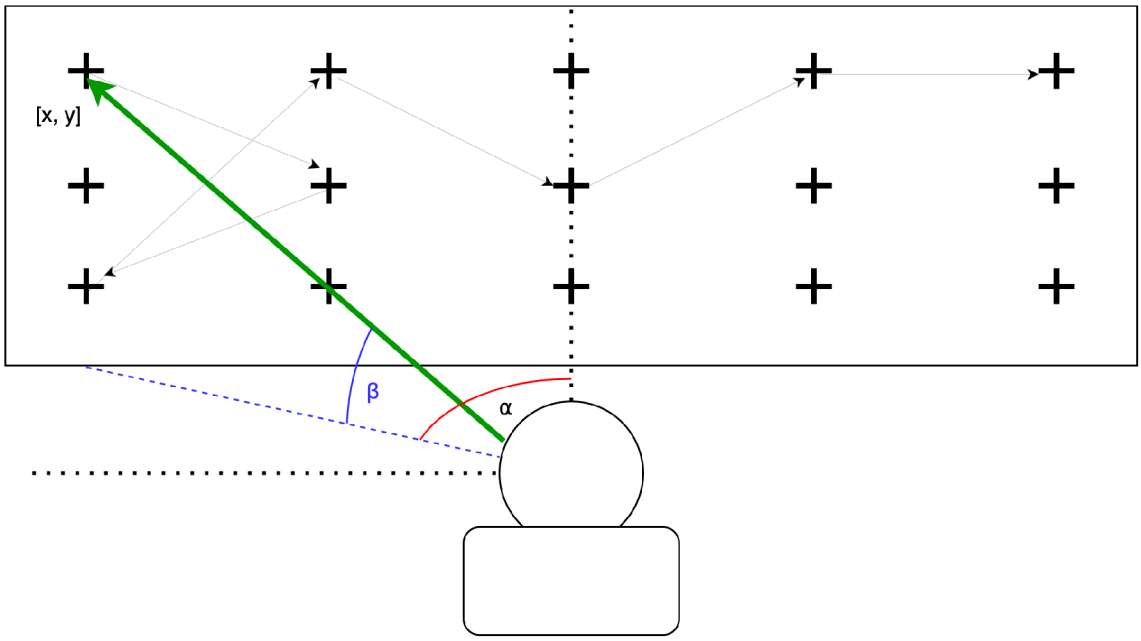
Většina metod pro měření směru pohledu je přesnější, pokud uživatel provede kalibraci. Kalibrací se rozumí výpočet či odhad tranformace směru pohledu, definovaného pomocí úhlů α a β, na souřadnice na monitoru [x, y].

Nejčastěji se zobrazuje mřížka bodů na obrazovce, které uživatel sleduje. Jsou známy souřadnice bodu [x, y] a z algoritmu získáme odpovídající data - v nejjednodušším případě úhly směru pohledu [α, β].

Cílem je zjistit funkce:

$$x_s = f(x_e, y_e), y_s = f(x_e, y_e) \tag{1.6}$$

Jedním ze způsobů, jak vypočítat transformaci souřadnic, je ortogonální Prokrustova analýza. Řeší způsob, jak vypočítat transformační matici, která co nejlépe mapuje množinu A do B pomocí ortogonální matice T tak, aby součet čtverců residuální matice $E = AT - B$ byl minimální. Je to lineární transformace zahrnující translaci, rotaci a uniformní změnu měřítka.



Obr. 1.19: Proces kalibrace měření směru pohledu

Matematicky je problém popsán jako:

$$AT = B + E, \tag{1.7}$$

$$TT' = T'T = I, \tag{1.8}$$

$$tr(E'E) = min, \tag{1.9}$$

Řeší se pomocí singulárního rozkladu (SVD). Tento přístup je vhodný, pokud jsou data z algoritmu měření směru pohledu lineárně rozložená. Uniformní změna měřítka neumožňuje nelineární transnformaci vstupních bodů.

Další ze způsobů je polynomická regrese pomocí polynomu druhého řádu.

$$\begin{aligned} x_s &= A_x x_e^2 + B_x y_e^2 + C_x x_e + D_x y_e + E_x \\ y_s &= A_y x_e^2 + B_y y_e^2 + C_y x_e + D_y y_e + E_y \end{aligned} \tag{1.10}$$

Koeficienty těchto rovnic lze nalézt pomocí levenberg marquardt algoritmu.

Podle autorů článku [7] je však nejlepší způsob pro transformaci souřadnic regrese pomocí SVR.

1.6.6 Hodnocení přesnosti algoritmů směru pohledu

Měření přesnosti algoritmu může být provedeno podobným způsobem jako kalibrace. Uživatel se dívá na podněty na obrazovce ve formě náhodně zobrazovaných bodů v mřížce a algoritmus analyzuje jeho snímky.

Nejčastěji se uvádí přesnost a někdy i opakovatelnost měření. Přesnost (accuracy) je definována jako průměrný rozdíl mezi detekovanou a reálnou pozicí kalibračního bodu. Rozdíl může být vyjádřen v px či mm pro konkrétní experiment, ale většinou se provede přepočet na chybu vyjádřenou v úhlu. Je vypočtena vzdálenost predikovaných (X_{pred}) a zobrazených bodů (X):

$$\Delta_i = \sqrt{(X_i - X_{ipred})^2} \tag{1.11}$$

$$accuracy = \frac{1}{n}\sum_{i=1}^{n}\Delta_i \tag{1.12}$$

Opakovatelnost (precision) je definována jako schopnost algoritmu spolehlivě reprodukovat stejné měření. Je vypočtena jako kvadratický průměr rozdílů a_i mezi měřeními stejného bodu.

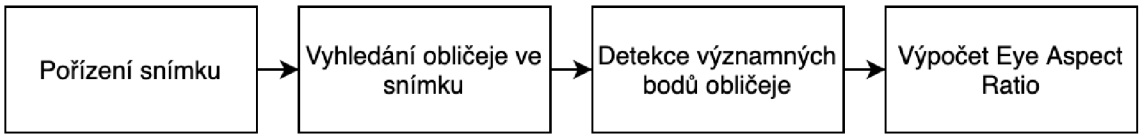
$$a_i = x_i - x_{i-1} \tag{1.13}$$

$$precision = \sqrt{\frac{1}{n}\sum_{i=1}^{n}a_i^2} \tag{1.14}$$

1.7 Detekce mrkání

1.7.1 Eye Aspect Ratio

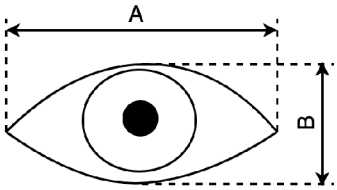
Jedním z přístupů pro detekci mrkání je výpočet Eye Aspect Ratio. Zpracování obrazu probíhá podle následujícího schématu:



Obr. 1.20: Algoritmus mrkání

Nejprve je snímek pořízen. Pomocí kaskády haarových filtrů nebo jiného detektoru obličeje je vyhledán obličej ve snímku. Výstupem detektoru jsou souřadnice a rozměry obdélníku, který ve snímku ohraničuje obličej. Poté jsou detekovány významné body obličeje. Z významných bodů oka je možné vypočítat, jak je oko otevřené - tzv. EAR (Eye Aspect Ratio). Spočítá se jako podíl výšky a šířky oka (B/A na obrázku 1.21).

Pokud je oko otevřené, poměr bude větší, pokud se oko zavře, rozměr B se výrazně zmenší a poměr bude malý. Nastaví se prahová hodnota, a pokud bude EAR menší než tato hodnota, oko je považováno za zavřené.



Obr. 1.21: Rozměry oka pro výpočet EAR

1.8 Existující implementace

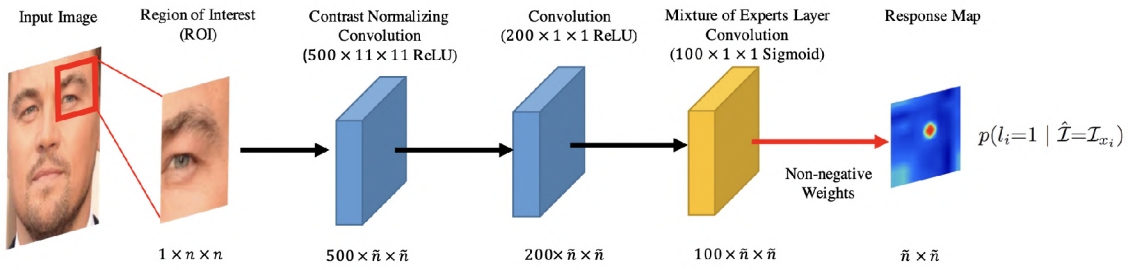
1.8.1 OpenFace 2.0

OpenFace 2.0 [2] je opensource toolkit pro analýzu snímků obličeje. Implementuje funkce:

- Detekce 3D významných bodů obličeje
- Odhad směru pohledu (Eye Gaze Estimation)
- Odhad natočení hlavy (Head Pose Estimation)
- Detekce akcí obličeje (Action Unit Recognition - úsměv, zvednutí obočí, mrkání, . . .)

Landmark detector

Pro detekci významných bodů obličeje se používá Convolutional Experts Constrained Local Model (CE-CLM). CE-CLM je vylepšení CLM, tato architektura přidává neuronovou síť Convolutional Experts Network. CEN vypočítá tzv. Response Map vstupu, lokální maxima Response map odpovídají významným bodům. Významné body jsou počítány nezávisle na sobě. V dalším kroku se provede aktualizace parametrů Point Distribution Modelu.



Obr. 1.22: Detekce pomocí Convolutional Experts Network [1]

Je robustní vůči natočení hlavy i vůči zakrytí určité části obličeje. Příklad detekce významných bodů obličeje tímto modelem je na obrázku 1.23

Obr. 1.23: Příklady detekce významných bodů pomocí CE-CLM modelu [1]

Detekce směru pohledu

Odhad směru pohledu je realizován pomocí Constrained Local Neural Field (CLNF) landmark detektoru [24].

Ten detekuje oční víčko, duhovku a zornici oka. Pro natrénování tohoto modelu byl použit dataset SynthesEyes[22]. U datasetů směru pohledu je často problém s přesnou anotací. Syntetický dataset tento problém eliminuje.

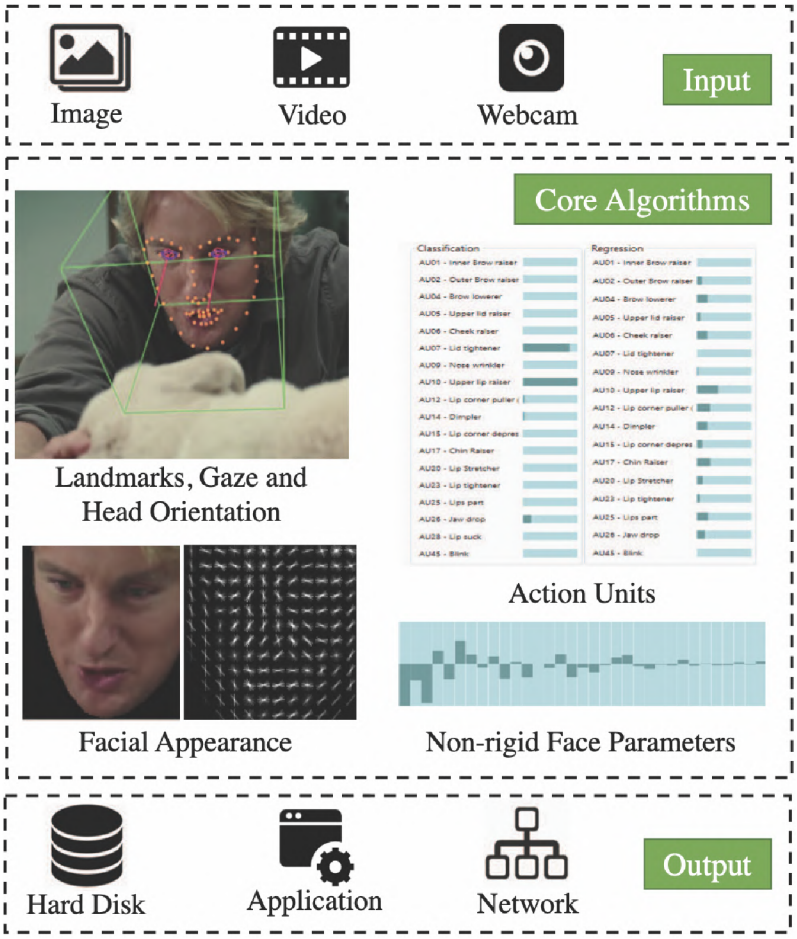
Byl vytvořen model oční bulvy včetně duhovky, rohovky a zornice. Poté byly pořízeny přesné 3D skeny několika subjektů různých věků, pohlaví a etnik. Oči subjektů byly nahrazeny modelem oční bulvy. Naskenovaná síť byla zjednodušena pomocí retopologie. Na obrázku 1.24 d) je dále vidět anotace zornice, duhovky a viditelné části oční bulvy. Anotace byla vytvořena ručně. Výsledný model má několik vstupních parametrů - pozice kamery, vektor směru pohledu, osvětlení, barva duhovky. Generuje snímky datasetu anotované významnými body.

Pro rozpoznání směru pohledu byl natrénován CLNF model. Pro trénink bylo vygenerováno 11382 snímků, rozptyl natočení hlavy byl 40° a směr pohledu 90°. Model dokáže odhadovat směr pohledu s průměrnou chybou cca 8-10° (cross dataset test, trénink pomocí SynthesEyes, validace na MPIIGaze)

Obr. 1.24: Postup od 3D skenu k modelu oka [24]

OpenFace lze použít jako aplikaci ve Windows, která analyzuje video z webkamery, zaznamenává uživatele a výstupy výše uvedených funkcí loguje do csv souboru.

Protože je celý projekt opensource, je možné použít výše uvedené funkce i v C++ projektu.



Obr. 1.25: OpenFace 2.0 [2]

1.8.2 iTracker & GazeCapture [12]

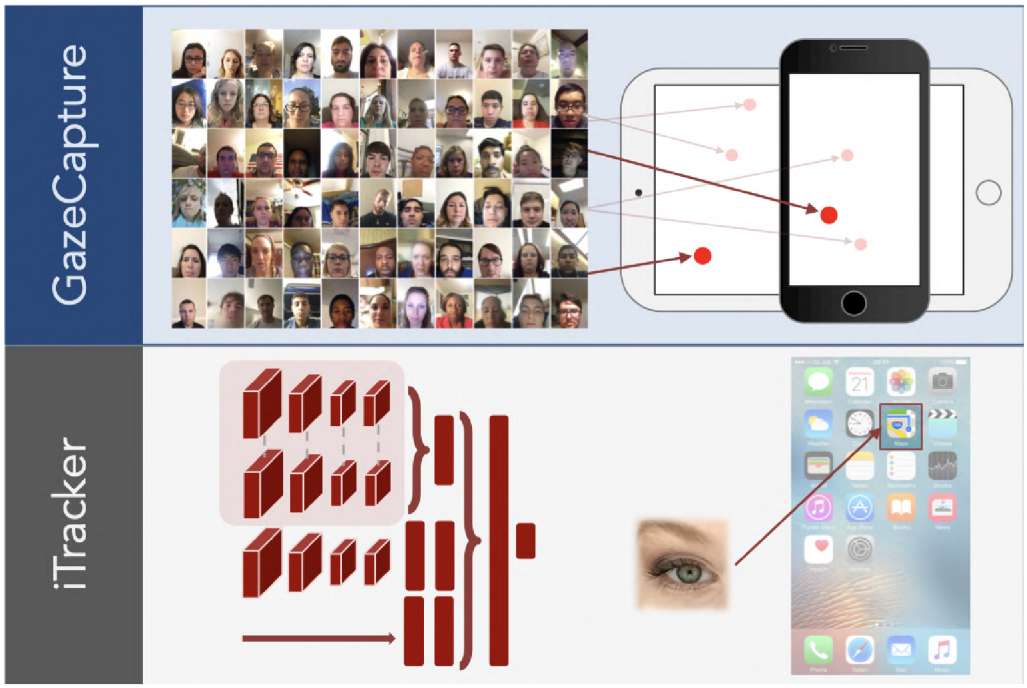
GazeCapture je dataset, který je první tzv. large scale dataset pro analýzu pohledu. iTracker je neuronová síť, která je natrénovaná na GazeCapture.

Dataset

Pro vytvoření datasetu autoři vyvinuli iOS aplikaci GazeCapture, která nahrávala snímky uživatelů dívajících se na definované místo na obrazovce mobilního telefonu či tabletu.

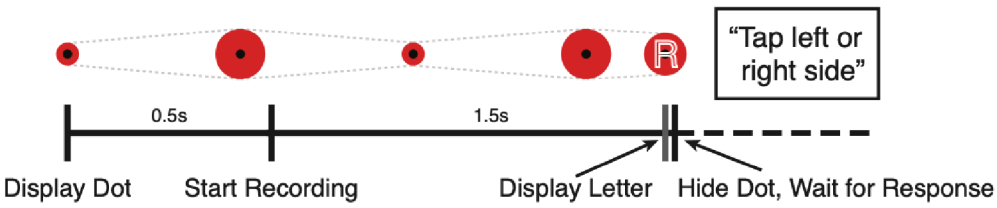
Nejjednodušší řešení by bylo uživatelům ukazovat body na náhodných místech na obrazovce, ale to by samo o sobě nemuselo být dostatečně robustní a spolehlivé.

Aby nebyli uživatelé vyrušeni notifikací, aplikace při nahrávání běží v Airplane módu bez připojení k internetu. Dále se pro zvýšení pozornosti uživatele používá pulzující červený kruh okolo černého bodu. Nahrávání uživatele je spuštěno 0.5 vteřiny po zobrazení bodu.



Obr. 1.26: iTracker a GazeCapture schéma [12]

Pozornost uživatele je kontrolována požadavkem na zmáčknutí levé či pravé strany obrazovky po každém bodu. Pokud se uživatel splete, musí bod zopakovat.



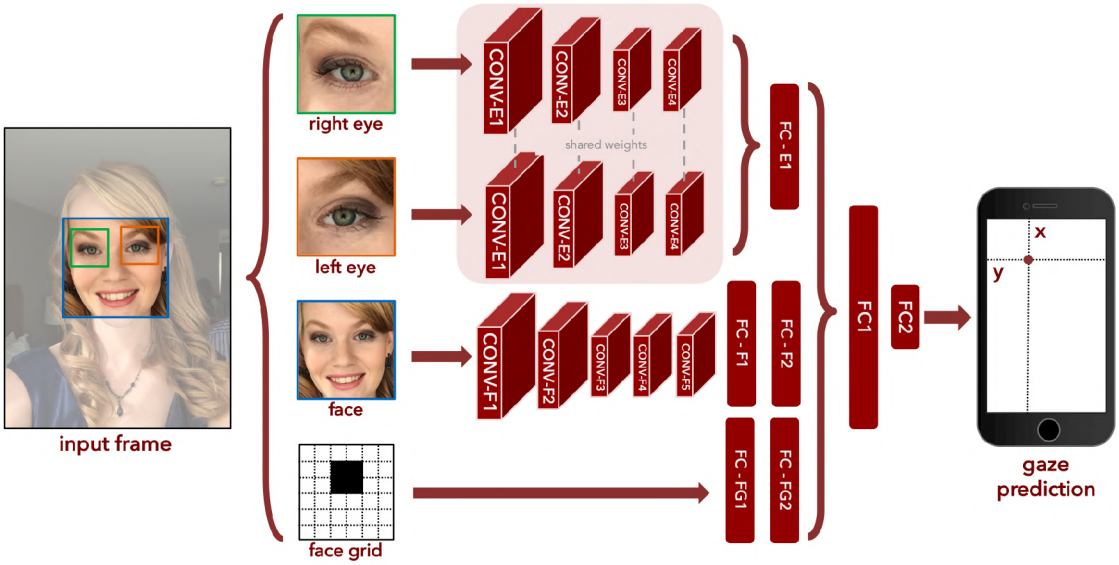
Obr. 1.27: Princip zobrazování bodu uživateli [12]

Nakonec je ještě ověřováno, že je na snímku z kamery vidět obličej pomocí real-time detektoru obličeje.

Dataset obsahuje 2445504 snímků od 1474 uživatelů.

iTracker

Je konvoluční neuronová síť se čtyřmi vstupy, jak je vidět na obrázku 1.28. Čtvrtý vstup, pojmenovaný face grid, je binární maska použitá pro určení pozice a velikosti obličeje vůči kameře. Vstupy pro snímky mají rozměr 225x225, face grid 25x25.



Obr. 1.28: Architektura iTracker [12]

Natrénovaný model je schopen predikovat přibližně 10-15 snímků za sekundu na mobilním telefonu. Chyba odhadu místa pohledu na obrazovce je 1.71 cm a 2.53 cm bez kalibrace, s kalibrací je chyba redukována na 1.34 cm a 2,12 cm.

1.8.3 GazeFlow

GazeFlow je eye-tracker software pro analýzu pohledu uživatele z webkamery. Umožňuje mimo jiné hýbat s myší pomocí pohledu. Software lze vyzkoušet na stránce `https://gazerecorder.com/app`. Hlavní využití má pro analýzu webových stránek, respektive v určování, do kterých míst se uživatel nejčastěji dívá.

Není blíže popsáno, jak tento software funguje. API je napsáno v javaskriptu, lze ho jednoduše implementovat na webovou stránku a využívat jeho funkce.

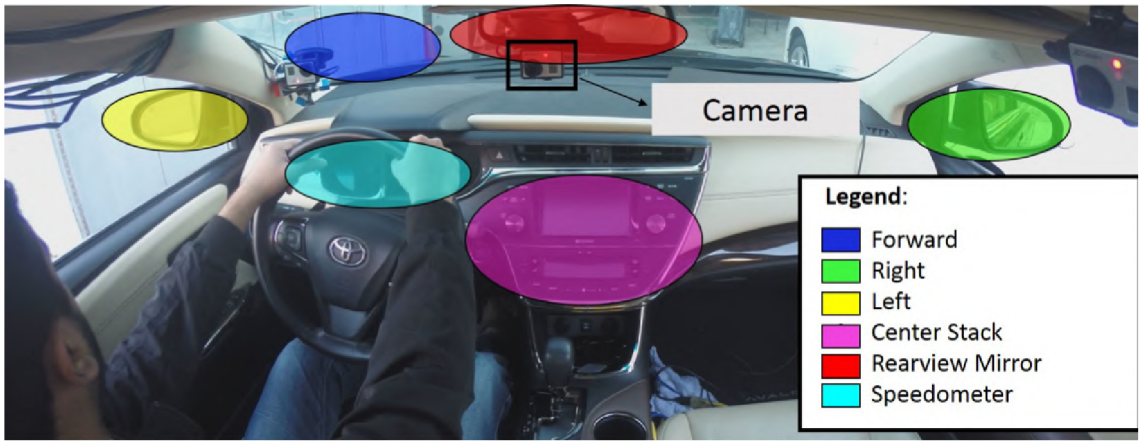
Obr. 1.29: Heatmapa pohledu na webovou stránku pomocí GazeFlow API

1.8.4 Estimace zóny pohledu řidiče pomocí Transfer Learning

Autoři práce [21] se nesnažili přesně určit směr pohledu, pohled řidiče rozdělili do šesti zón (obrázek 1.30)

1. Pohled dopředu
2. Pohled do pravého zrcátka
3. Pohled do levého zrcátka
4. Pohled na infotainment
5. Pohled do středového zrcátka
6. Pohled na tachometr

Jak je vidět na obrázku, hlavní kamera je umístěna blízko středového zpětného zrcátka.



Obr. 1.30: Zóny pohledu řidiče [21]

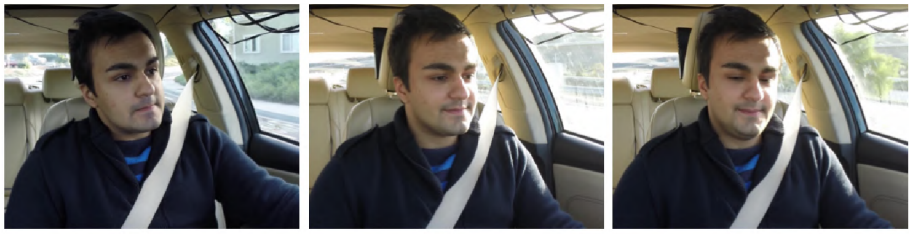
Dataset

Dataset byl vytvořen s pomocí deseti lidí, kteří řídili dvě různá auta v reálném provozu. Uvnitř vozidla se kromě kamery u středového zrcátka řidiče nacházely ještě dvě další kamery - záběry z nich pomohly při ručním anotování snímků.

Celkem použitý dataset obsahoval 47515 anotovaných snímků, největší podíl (21522 snímků) měly snímky pohledu dopředu. Pro ostatní zóny bylo přibližně 4000 snímků na zónu.

CNN architektura

Pro detekci zóny bylo použito principu transfer learning. Pro tento účel byly použity architektury AlexNet, VGG16, ResNet50 a SqueezeNet. Bylo zjištěno, že nejpřesnější je SqueezeNet s přesností 95.18%. Zároveň bylo ukázáno, že není třeba extrahovat obličej z obrazu pomocí detektoru, protože neuronová síť dokáže sama oči najít.



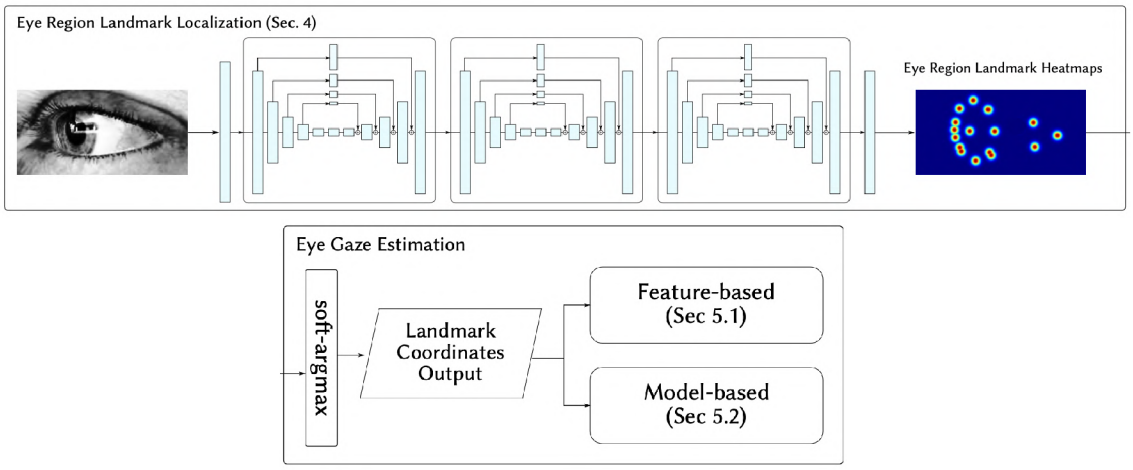
Obr. 1.31: Příklady obrázků v datasetu [21]

1.8.5 GazeML

GazeML [15] využívá k určení bodů oka tzv. Stacked Hourglass network, což je několik CNN typu autoenkodér za sebou. Síť je natrénovaná pouze s pomocí syntetických dat z datasetu UnityEyes. Směr pohledu je určen dvěma způsoby.

Model-based přístup se snaží estimovat parametry 3D modelu oka uživatele. Mezi ně patří odhad středu a poloměru oční bulvy a středu zornice. Tímto způsobem na EYEDIAP datasetu je dosaženo průměrné chyby 11.9°.

Druhým je feature-based způsob, kde je natrénován SVR model na extrahovaných příznacích. Vstupem je 17 normalizovaných souřadnic významných bodů a prior (počáteční odhad směru pohledu). Feature based přístup má na EYEDIAP datasetu průměrnou chybu 7.4°.



Obr. 1.32: Schéma architektury GazeML [15]

2 Simulátor řízení

Nejdůležitější částí celého projektu je simulátor řízení. Od jeho vlastností, parametrů, požadavků se bude odvíjet navržené řešení pro detekci pohledu a mrkání, potažmo únavy. Nejprve tedy popíšu celý systém a jeho vlastnosti.

Obr. 2.1: Pohled na simulátor řízení

2.1 Hardware simulátoru

Základem simulátoru je ultra-širokoúhlý, zakřivený monitor Samsung CHG90 QLED o úhlopříčce 49 palců, který zobrazuje v rozlišení 3860x1080 pixelů. Radius zakřivení monitoru je 1,8 m.

Uživatel simulátoru sedí ve skořepinové sedačce, obličej je vzdálen od monitoru 60-80 cm. Simulátor ovládá pomocí herního volantu s pedály a řadicí pákou.

Jádrem systému je počítač s 12 jádrovým procesorem AMD Ryzen 9 3900XT, grafickou kartou NVIDIA GeForce RTX 2080, 32 Gb operační paměti a SSD diskem 512 Gb. Výpočetního výkonu je tedy dostatek.

2.1.1 Kamera

Ve výchozím stavu simulátor není vybaven kamerou, která by sledovala řidiče.

Požadavky na kameru jsou:

- Rozlišení alespoň 720p
- Snímkovací frekvence minimálně 30 snímků/s
- Ostření na vzdálenost 60 - 80 cm
- Zorné pole $> 40°$
- Připojení pomocí USB

Pro první testování na simulátoru byla použita kamera Logitech StreamCam C980, která už byla na ústavu automatizace zakoupena.

Dokáže natáčet sekvence snímků v rozlišení 1080p se snímkovací frekvencí 60 snímků za sekundu. Připojena je do počítače pomocí USB 3.2. Objektiv má světelnost f/2.0 a ohniskovou vzdálenost 3.7 mm. Dokáže zaostřit na vzdálenosti 10 cm až nekonečno. Zorné pole kamery je 78°. Je vybavena držákem, který umožňuje snadné uchycení na stativ či monitor.

Obr. 2.2: Kamera Logitech StreamCam C980 `https://www.logitech.com/`

Na obrázku 2.3 je vidět jeden ze snímků datasetu v rozlišení 1920x1080 px a přiblížený detail obličeje a oka. Snímek má dobrou kvalitu, zorné pole je možná až zbytečně velké. I tak je po přiblížení detail obličeje poměrně kvalitní. Obličej má rozlišení 350x500 px, oko 90x40 px. Kamera tedy splňuje požadavky, je vidět že snímky jsou ostré a dostatečně detailní. Proto byla využita i do finálního řešení.

2.2 Software simulátoru

Software pro simulátor řízení je vytvořen pomocí Unreal Engine. Unreal Engine byl představen v roce 1998 pro tvorbu akčních počítačových her. Nyní je využíván pro všechny typy počítačových her, dále se používá i ve filmovém průmyslu.

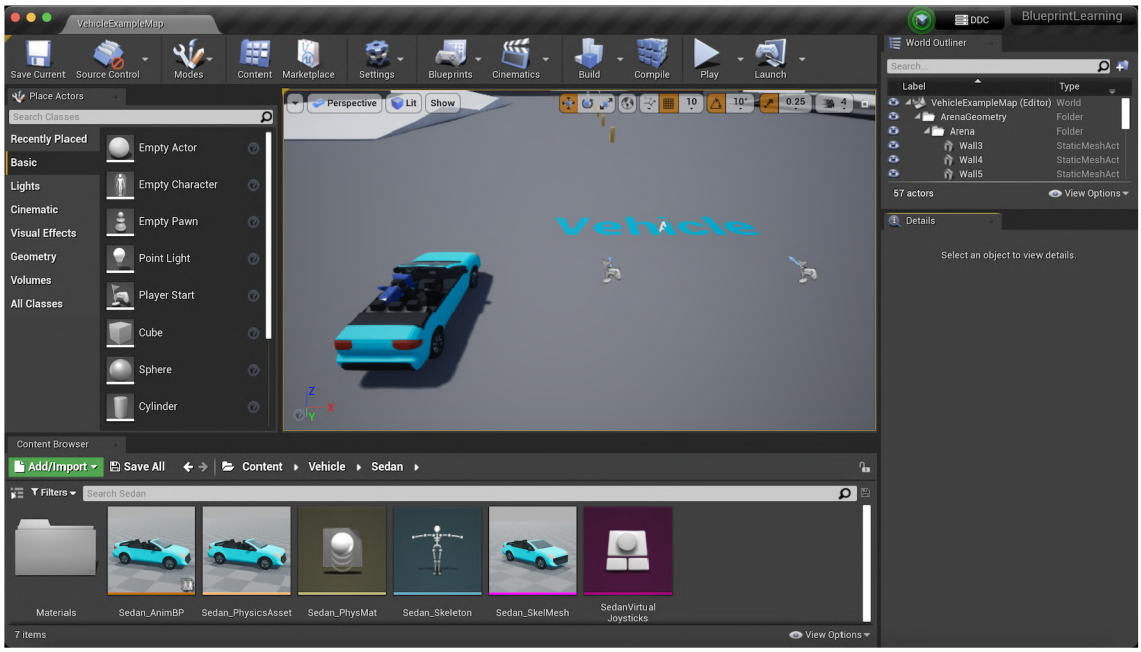
Obr. 2.3: Snímek z kamery včetně detailu obličeje a oka

2.2.1 Unreal Engine

(Informace o UE z `https://docs.unrealengine.com/`) Unreal Engine je vývojové prostředí, ve kterém je možné navrhnout grafické prostředí i logiku hry. Návrh grafického prostředí hry lze tvořit v editoru. Do každé úrovně (levelu) hry je možné přidávat různé objekty, tvořit krajinu a podobně. Pokud budeme chtít přidat do scény krychli, lze ji jednodušše přetáhnout z levé nabídky objektů do prostředí hry a poté s ní můžeme pohybovat, otáčet, zvětšovat apod. Dále můžeme nastavovat texturu a přiřazovat k objektu různé funkce či třídy.

V horní nabídce editoru jsou důležitá tlačítka Compile a Play. Compile slouží pro kompilaci blueprintů a kódu v C++. Play spustí zrovna otevřený level, tímto způsobem je tedy možné hru testovat. Pokud je hra hotová, lze ji se všemi závislostmi sestavit pomocí příkazu Package. Tímto způsobem se hra zabalí pro nastavenou platformu a poté ji můžeme spustit pomocí souboru .exe. Detailnější popis programování pomocí UE bude v kapitole 4.

V případě software simulátoru Unreal Engine poskytuje funkce, které generují



Obr. 2.4: Unreal Editor - grafické prostředí

grafické prostředí celého simulátoru. S jeho pomocí jsou vytvořeny prostředí jednotlivých úrovní, které vždy obsahují vozovku, auto a okolní krajinu. V každé úrovni jsou různé scénáře, specifické pro každé měření. S pomocí UE lze také přidávat různé úhly pohledu z kamer či menu pro navigaci ve hře. Software simulátoru je detailněji popsán v kapitole 4.1.

2.3 Požadavky na detekci směru pohledu a mrkání

Algoritmus má být použit pro zjištění, zda uživatel při ovládání vozidla sleduje vozovku a věnuje se řízení. Požadovaná přesnost určení pohledu na obrazovce je proto poměrně malá. Šířka monitoru je 1200 cm, bod pohledu stačí určit s přesností 10 cm. Při vzdálenosti od monitoru 60-80 cm to přibližně odpovídá úhlu 7-10°.

Další požadovanou funkcí je detekce mrkání. Detekce mrkání by měla fungovat bez ohledu na to, kam se uživatel dívá. Po dokončení měření by měl algoritmus určit frekvenci mrkání (počet za minutu) a vypočítat délku každého mrknutí.

Obr. 2.5: Obrazovka simulátoru s vyznačenou tolerancí ± 10 cm

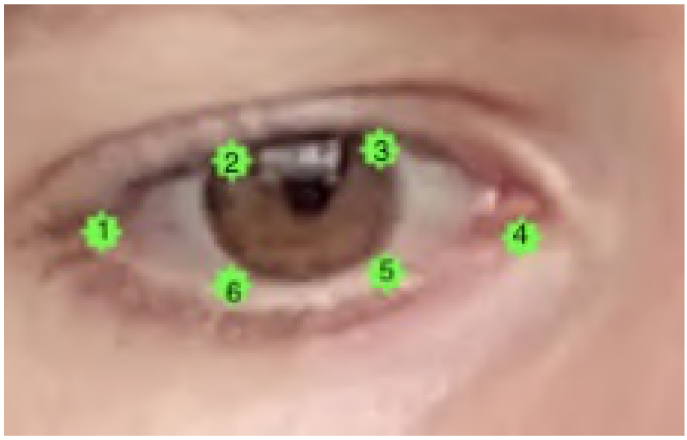
3 Otestované algoritmy

Své pokusy s neuronovými sítěmi jsem realizoval v jazyce Python. Výsledný algoritmus však musí být v C++, protože to je jazyk, ve kterém funguje Unreal Engine. Python je sice pomalejší než C++, pro zpracování obrazu a machine learning je však mnohem více populární. V případě chyb lze snadněji vyhledat řešení. Plán byl takový, že nejprve otestuji různé metody co nejjednodušším způsobem a nejlepší přístup pak implementuji do systému simulátoru v C++.

Pro implementaci algoritmů zpracování obrazu byla využita knihovna OpenCV, tvorba a trénink CNN byl realizován pomocí Keras modulu Tensorflow. V rámci vývoje algoritmů jsem pro test používal webkameru notebooku, ze které jsem načítal snímky ve while smyčce a ty analyzoval. Tímto způsobem jsem mohl algoritmy jednoduše testovat.

3.1 Detekce mrkání pomocí výpočtu EAR

Výpočet EAR byl implementován podle schématu 1.20. Pomocí cv::VideoCapture je pořízen snímek z kamery. Poté je převeden na šedotónový, aby bylo zpracování obrazu rychlejší. Jako detektor obličeje je použita Haarova kaskáda. Pro detekci významných bodů obličeje byl použit cv::FacemarkLBF (lbfmodel.yaml). Tento algoritmus určuje významné body s pomocí regresních stromů, které jsou naučeny na datasetech LFPW, Helen a 300W. Každé oko je reprezentováno šesti body (obrázek 3.1).



Obr. 3.1: Významné body oka

EAR se vypočítá jako:

$$EAR = \frac{||p2 - p6|| + ||p3 - p5||}{2 \cdot ||p1 - p4||} \tag{3.1}$$

Práh pro detekci zavřeného oka byl nastaven experimentálně na hodnotu 0,22.

Ve while smyčce je pro každý snímek z kamery zjištěno EAR pravého a levého oka uživatele. Pokud je EAR menší než 0,22, oko je považováno za zavřené. Skript počítá čas, dokud EAR není větší než 0,22.

Nevýhodou přístupu je velká závislost na landmark detektoru, který nemusí správně fungovat v situacích, kdy je hlava hodně natočená vůči optické ose kamery.

3.1.1 Zjištění frekvence a délky mrkání

Určení délky mrknutí je v tomto případě zjištění, jak dlouho je oko zavřené. Převedeno na výpočet EAR - jak dlouho je hodnota EAR menší, než 0,22.

Skript běží jako nekonečná while smyčka, ve které je měřen aktuální čas. Pokud EAR klesne pod 0,22, začne se do proměnné načítat čas. Pokud je v dalším snímku EAR vyšší než 0,22, načítání času se ukončí.

Frekvence mrkání je měřena tak, že se počítá počet mrkání za jednu minutu. Přesnost určení frekvence a délky mrkání je závislá na frekvenci snímkování.

3.2 Detekce natočení hlavy

Implementace detekce natočení hlavy byla realizována s pomocí návodu na stránce `learnopencv.com`. Detekce je zde realizována pomocí významných bodů obličeje. Pro detekci tímto způsobem potřebujeme:

- 2D souřadnice významných bodů obličeje (ze snímku)
- 3D souřadnice těch stejných bodů (z modelu obličeje)
- Intristické parametry kamery

2D souřadnice obličeje

Pro detekci 2D významných bodů obličeje je možné použít stejný landmark detektor jako v předchozí kapitole. V návodu jsou použity body - špička nosu, okraje úst, brada a okraje očí.

3D souřadnice z modelu obličeje

Nelze jednoduše získat 3D souřadnice bodů obličeje, proto je obličej aproximován 3D modelem.

Intristické parametry kamery

Aby algoritmus fungoval, je třeba znát parametry kamery. Mezi ně patří ohnisková vzdálenost kamery, střed optické soustavy a zkreslení. I tyto parametry lze aproximovat - střed optické soustavy bude střed snímku, ohnisková vzdálenost šířka obrazu v pixelech a zkreslení nebudeme předpokládat.

OpenCV obsahuje pro tzv. Pose Estimation funkci solvePNP.

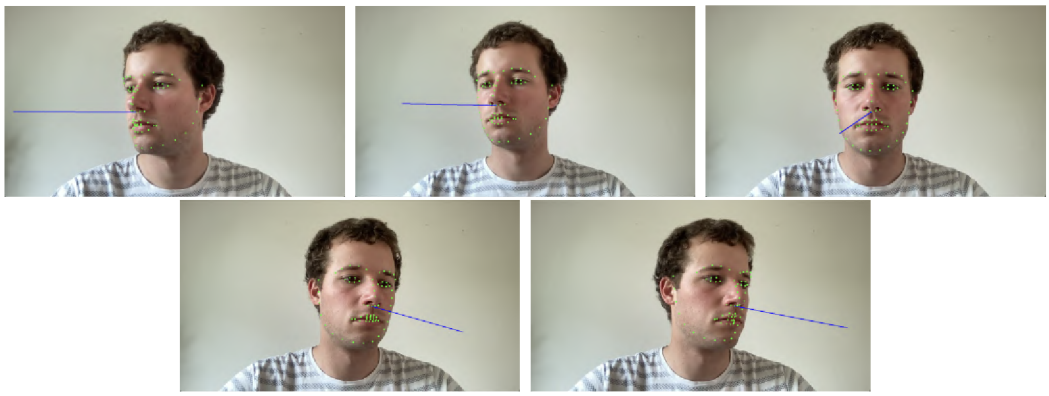
Parametry funkce jsou:

- objectPoints - vektor 3D souřadnic obličeje
- imagePoints - vektor odpovídajících 2D souřadnic ve snímku
- cameraMatrix - matice parametrů kamery
- distCoeffs - vektor koeficientů zkreslení
- rvec - Výstupní rotační vektor
- tvec - Výstupní translační vektor

Snímek po analýze natočení hlavy je na obrázku 3.2. Vybral jsem pět snímků, které zobrazují různé horizontální natočení hlavy v celém rozsahu. Při větším natočení hlavy už použitý detektor nedokáže významné body obličeje najít.

Na detailu snímku 3.3 je vidět, že detektor v levé části obrázku není moc přesný, což má za následek i méně přesný odhad směru natočení hlavy.

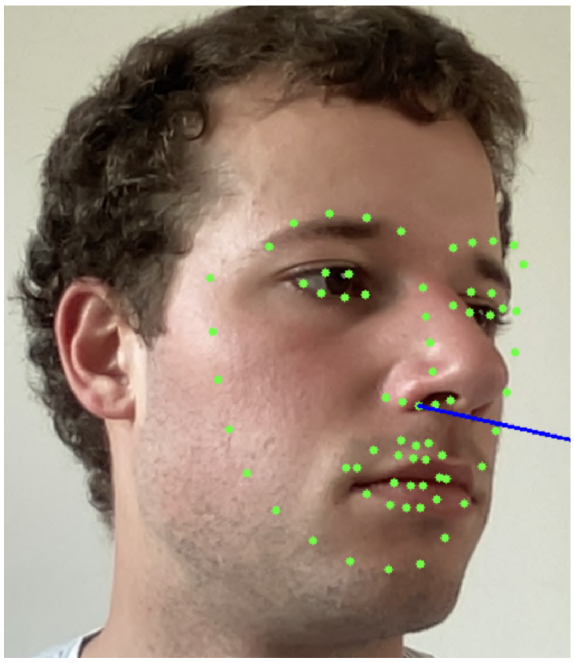
Zároveň je zřejmé, že ačkoliv natočení hlavy může být podobné se směrem pohledu, v mnoha případech tomu tak není a je nutné směr pohledu zjišťovat z očí.



Obr. 3.2: Zobrazení směru natočení hlavy

3.2.1 Detekce středu oka

Pro detekci pohledu je možné detekovat pozici středu zornice vůči okrajům oka. Jednou z možností, jak detekovat střed zornice, je využití prahování a morfologických operací. Nejprve je ze snímku s pomocí landmarků extrahována pouze část oka. Poté je aplikován bilaterální filtr. Dále je provedena operace erode a nakonec se použije



Obr. 3.3: Chybná detekce významných bodů při velkém natočení hlavy

binární prahování. Na naprahovaném snímku jsou vyhledány kontury a střed oka je určen jako centroid oblasti, kterou kontura ohraničuje. Postup je ukázán na obrázku 3.4.

Obr. 3.4: Postup hledání středu oka prahováním

Tento přístup jsem našel implementovaný v Pythonu na Github - `https://github.com/antoinelame/GazeTracking`. Detekce funguje v případě tmavých očí velmi dobře, problém je ve chvíli, kdy není velký rozdíl v barvě duhovky a sklivce, například u modrých očí. Zde algoritmus občas hůře zvládá oko naprahovat a střed je pak určen nepřesně.

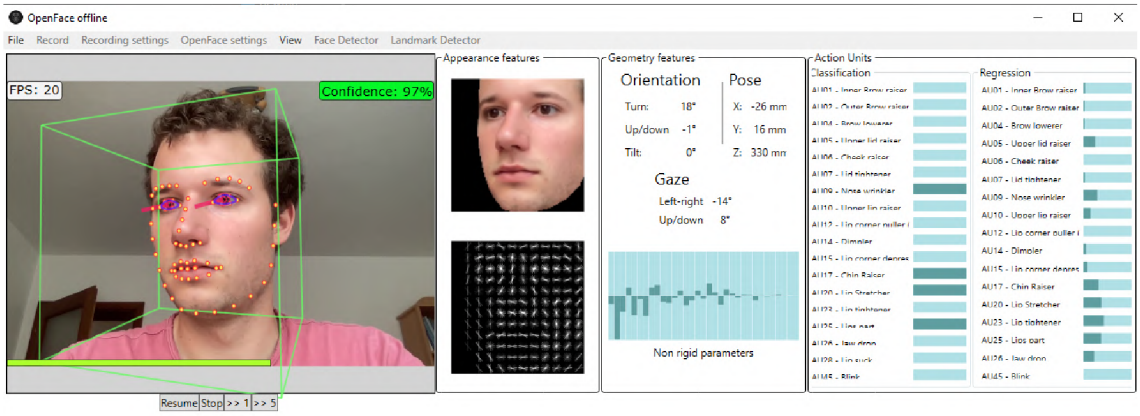
Také jsem zjišťoval, zda by bylo možné pro střed oka použít vyhledávání kružnic pomocí Houghovy transformace. Bohužel se mi tento způsob nepodařilo implementovat tak, aby byla detekce spolehlivá.

3.3 Implementace OpenFace 2.0

Jak již bylo uvedeno, OpenFace je knihovna, s jejíž pomocí by mělo být velice jednoduché vyřešit v podstatě všechny body zadání této práce. Repozitář na github.com

obsahuje vše potřebné ke zkompilování knihovny. Na Windows jsou ke stažení již zkompilované soubory knihovny.

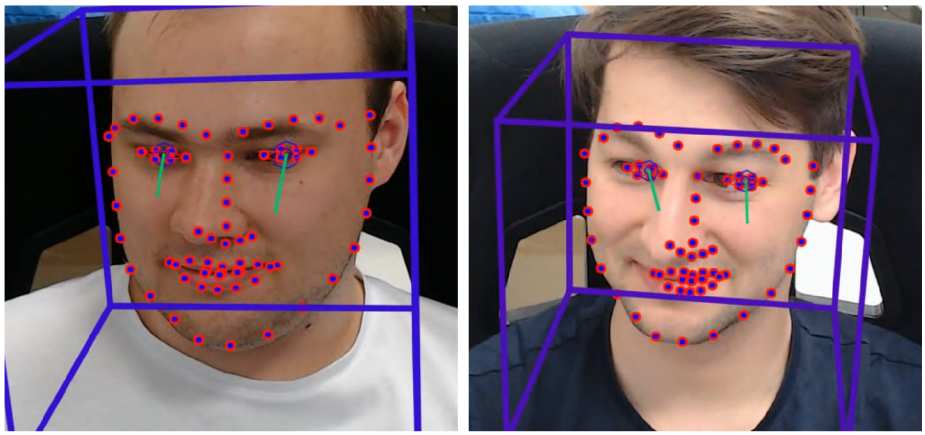
OpenFace zároveň má i grafické prostředí, které lze využít pro otestování funkcí knihovny s jakoukoliv kamerou připojenou do počítače. Prostředí je na obrázku 3.5.



Obr. 3.5: OpenFace GUI

Pomocí GUI jsem otestoval detekci směru pohledu i detekci mrkání na simulátoru. Zvládá detekovat směr pohledu na první pohled velice spolehlivě. Natočení hlavy odhaduje bez problémů, použitý landmark detektor je velmi stabilní a robustní. Odhad směru pohledu také funguje dobře, nicméně při velkém natočení očí začíná být problém s přesnou detekcí. Je otázka, do jaké míry je to problém, protože většinou člověk otočí hlavu, když se má dívat tak do strany. Dva nepovedené odhady směru pohledu jsou na obrázku 3.6.

Program zaznamenává csv soubor všech analyzovaných informací z obličeje společně s časovou značkou, takže je teoreticky možné nechat tento program běžet na pozadí simulátoru a zaznamenaná data analyzovat skriptem ex post.



Obr. 3.6: Chybné detekce pomocí OpenFace

Mým cílem však bylo použít OpenFace jako C++ knihovnu. Zpočátku jsem nevěděl jak na to, ale bližším zkoumání jsem pochopil, jak celý systém Openface funguje. Základem je projekt ve Visual Studiu OpenFace.sln. Obsahuje několik podprojektů:

- FeatureExtraction - command line program pro analýzu snímků
- FaceAnalysis - statická knihovna pro analýzu obličeje (tzv. Action Units)
- GazeAnalysis - statická knihovna pro analýzu směru pohledu
- LandmarkDetector - statická knihovna pro určení významných bodů obličeje

Po zkompilování knihoven vzniknou soubory .lib. Knihovny je možné staticky linkovat a využít v jakémkoliv C++ projektu. Pro správnou funkci je ještě potřeba knihoven OpenCV, dlib a OpenBLAS.

Protože OpenFace je knihovna, kterou jsem se rozhodl použít pro finální řešení, není zde detailní testování funkcí směru pohledu a mrkání. Tato část bude v kapitole kalibrace a testování měření 4.4.

3.4 Analýza směru pohledu pomocí CNN

Jedním z prvních pokusů bylo vytvoření a trénink neuronové sítě pro odhad směru pohledu. Po přečtení článků o iTracker a GazeCapture [12] jsem zkusil takovou síť vytvořit a natrénovat pro odhad směru pohledu.

Pro trénování a tvoření modelu jsem používal Keras, což je knihovna pro strojové učení obsažená v Tensorflow.

3.4.1 Dataset

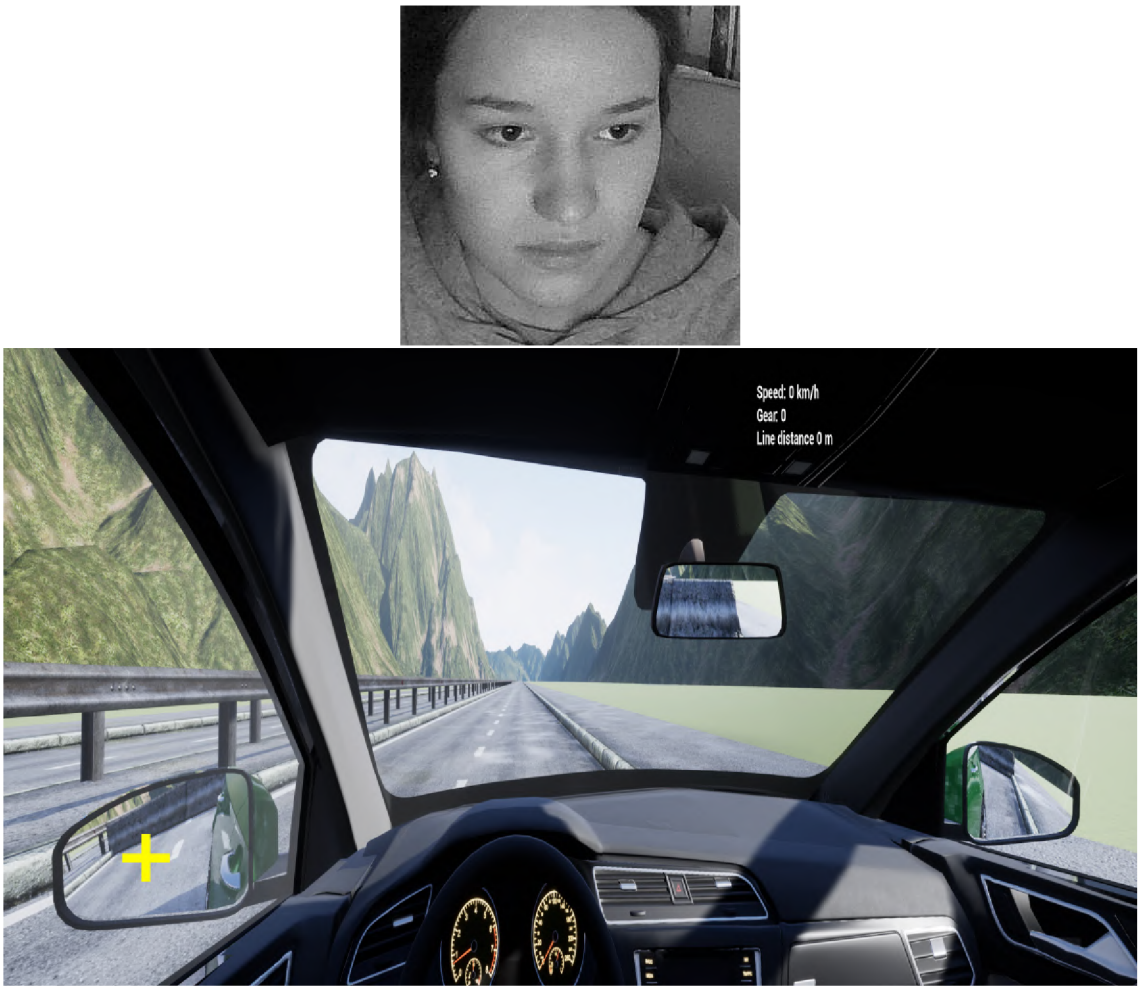
Pro trénink neuronové sítě odhadující směr pohledu bylo třeba vytvořit databázi snímků. Při trénování konvolučních neuronových sítí je dobrý dataset naprosto zásadní pro správnou funkci modelu.

Cílem algoritmu nejprve bylo určení souřadnic X a Y v rámci souřadnicového systému monitoru. Proto jsem každý snímek anotoval těmito souřadnicemi.

Skript pro tvorbu datasetu

Uživatel měl za úkol pohybovat kurzorem a zároveň jej sledovat na obrazovce simulátoru. Snímky z kamery byly zaznamenány pomocí skriptu createDataset.py, který neustále ukládal snímky obličeje uživatele, extrahované z celého záběru kamery pomocí haarovy kaskády.

Anotace snímku je zapsána do jeho názvu. Funkce, která je zavolána při pohybu myši v okně, dostane souřadnice x a y myši a snímek uloží pod názvem `x_y.jpg`. Příklad takového snímku je na následujícím obrázku s lokalizací x y v rámci okna simulátoru.



Obr. 3.7: Snímek z datasetu s vyznačením pohledu na obrazovce

Tímto způsobem byl vytvořen dataset o 9035 snímcích. Dataset tvořilo pět členů mojí rodiny. Protože se jednalo o první test, dataset nebyl vytvořen na simulátoru, ale doma na 28 palcovém 4K monitoru.

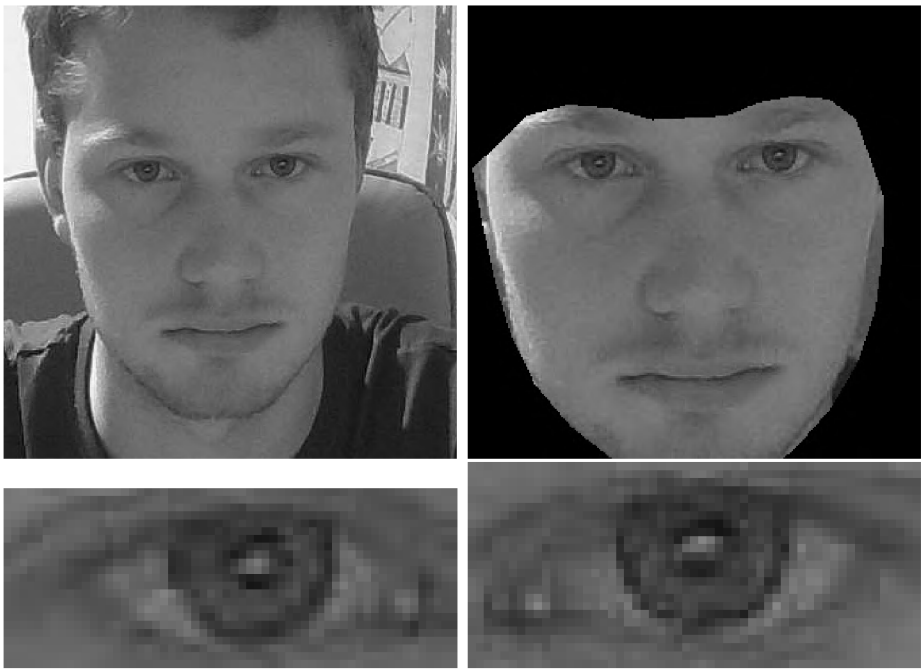
3.4.2 Preprocessing snímků

Před tím, než je poslán snímek do neuronové sítě, je nutné ho upravit tak, aby důležité prvky byly zvýrazněny a nepotřebné informace odstraněny. Mým cílem tedy bylo ve snímku lokalizovat obličej, oříznout nepotřebnou část snímku, odstranit pozadí, převést snímek na šedotónový a změnit jeho velikost tak, aby rozměr odpovídal vstupu neuronové sítě.

Obličej jsem lokalizoval pomocí haarovy kaskády. Pozadí za obličejem bylo vymazáno tak, že se vytvořila křivka ohraničující obličej z detekovaných okrajových významných bodů.

Nejprve jsem testoval jednoduchou neuronovou síť s jedním vstupem. V článku

iTracker [12] však používali neuronovou síť se čtyřmi vstupy pro celý obličej, levé oko, pravé oko a face grid. Preprocesing snímku v tomto případě také zahrnoval vyříznutí snímku levého a pravého oka s pomocí landmarků oka. Čtvrtý vstup - tzv. face grid pro odhad vzdálenosti obličeje od monitoru jsem neimplementoval pro zjednodušení algoritmu.



Obr. 3.8: Preprocessing

3.4.3 Architektura CNN

Při tvorbě CNN jsem se inspiroval od již existujících architektur.

Používal jsem velice podobnou architekturu neuronové sítě, jako používá algoritmus iTracker. Je to CNN se čtyřmi vstupy - snímek obličeje, levého a pravého oka 1.28.

3.4.4 Testování modelů

Přesnost modelu byla zjišťována pomocí rozdělení datasetu na trénovací a validační část. Dále jsem vždy model testoval pomocí skriptu, kde model predikoval místo pohledu na obrazovce v reálném čase ze snímků mého obličeje. V případě regresní CNN skript vykresluje bod do souřadnic predikovaných modelem, u CNN s klasifikací do pěti tříd byla predikce zobrazována do daného místa (při pohledu na levé zrcátko se v něm zobrazí bod).

Tímto způsobem bylo možné rychle získat představu, jak dobře model funguje v reálných podmínkách.

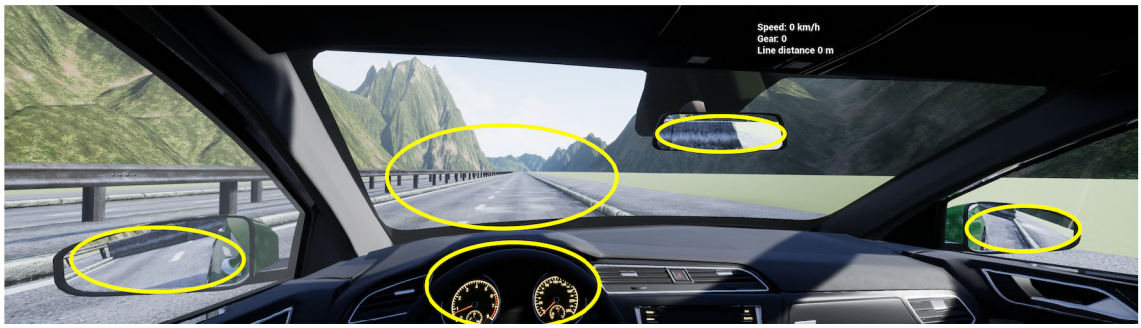
Regresní CNN

Nejříve jsem se domníval, že budu schopen natrénovat model tak, aby predikoval souřadnice, kam se uživatel v rámci obrazovky dívá. Pro tuto úlohu jsem chtěl vytvořit regresní konvoluční neuronovou síť. Tato cesta se po několika pokusech ukázala jako velmi složitá. Detekce přesného místa pohledu byla nespolehlivá, model se mi nedařilo správně natrénovat.

Došel jsem k závěru, že pro natrénování této sítě bych potřeboval mnohem větší dataset, než jsem byl schopen vytvořit. iTracker používali k natrénování dataset GazeCapture, který dohromady obsahuje přibližně 2.5 milionu obrázků od 1474 lidí, můj dataset obsahoval 9 tisíc obrázků od pěti lidí.

Klasifikační CNN

Protože není nutné přesně detekovat souřadnice pohledu, změnil jsem po neúspěchu s regresí souřadnic architekturu CNN. Neuronovou síť jsem upravil tak, aby klasifikovala do pěti tříd (vozovka, 3 zpětná zrcátka, tachometr). Třídy jsou na obrázku 3.9.

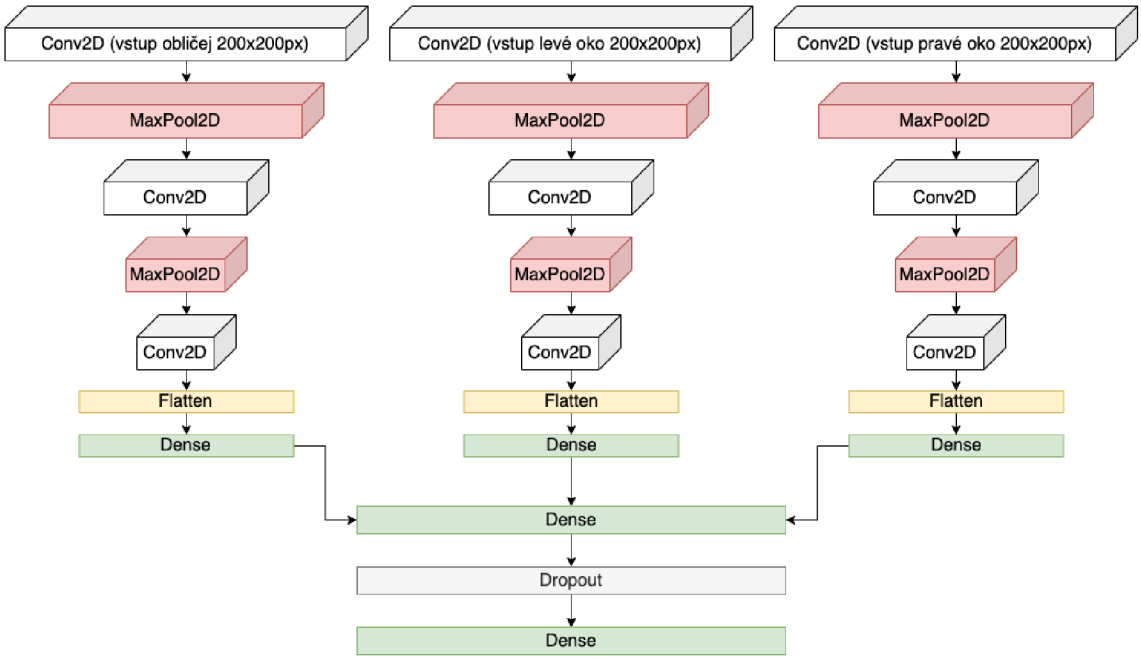


Obr. 3.9: Zóny detekované pomocí klasifikační CNN

Architektura sítě je na obrázku 3.10. Při trénování a testování této sítě jsem vybral pouze ty snímky, které odpovídaly pohledu do jedné z pěti oblastí, ostatní snímky jsem nepoužil. Počet snímků pro trénink rozdělený do kategorií je v tabulce 3.1. Pro validaci bylo náhodně vybráno 1000 snímků.

Levé zrcátko	Pravé zrcátko	Středové zrcátko	silnice	tachometr	Celkem
1330	1288	1368	1511	1495	5992

Tab. 3.1: Počty snímků v jednotlivých kategoriích



Obr. 3.10: Architektura použité CNN

Trénink sítě proběhl ve 20 epochách, ztrátová funkce byla nastavena na sparse categorical crossentropy, optimizer nastaven na Adam s learning rate 0.001. Natrénovaná síť dosahuje přesnosti 98% na validační množině.

Predikce modelu jsem dále otestoval s pomocí skriptu a webkamery. Model funguje dobře, dokáže pohled správně odhadovat. Je citlivý na natočení hlavy a zároveň na pohled pouze očima. Nefunguje však ve chvíli, kdy je obličej moc daleko či blízko od kamery. Architektura iTracker toto řeší dalším vstupem - částí neuronové sítě (face grid - obrázek 1.28), která odhaduje vzdálenost uživatele od kamery. Abych vliv vzdálenosti obličeje od kamery mohl zanedbat, použil jsem opěrku na bradu, aby ten, kdo vytvářel dataset či testoval model, byl vždy od kamery stejně daleko od monitoru.

Dále jsem testoval vliv různého osvětlení v místnosti. Dataset byl vytvořen záměrně v různých osvětleních, při testování jsem nepozoroval výraznější vliv změny osvětlení na kvalitu predikce.

Otestoval jsem i osobu, co dataset netvořila. Přesnost modelu se pro neznámé osoby pohybovala okolo 50-80 %. Generalizace CNN byla poměrně špatná, myslím si, že by to vyřešil dataset, který by vytvořilo více osob.

3.4.5 Využití SqueezeNet pro klasifikaci zóny pohledu

Podle článku [21] jsem se rozhodl, že vyzkouším transfer learning pro klasifikaci zóny pohledu. Využil jsem Matlab, který má SqueezeNet nadefinovanou v Deep Learning

Toolboxu, dále obsahuje i příklad .mlx, kde je detailně popsáno jak postupovat. Principem je použít nadefinovanou architekturu neuronové sítě, která je naučená na obrazová data. Poslední vrstvy sítě se nahradí tak, aby měla síť požadovaný počet výstupních tříd. Poté je třeba natrénovat parametry posledních vrstev sítě.

Také jsem upravil program pro vytváření datasetu tak, aby uživatel nemusel pohybovat kurzorem po obrazovce a zároveň jej sledovat. Místo toho byl zobrazován bod na obrazovce simulátoru vždy uprostřed konkrétní zóny. Snímky se ukládaly podle kategorie do složek.

SqueezeNet se mi ale nepodařilo natrénovat. Trénování bylo na mém počítači pomalé a nejvíce jsem se dostal na přesnost cca 60%.

3.5 Volba algoritmu

Po vyzkoušení různých přístupů k řešení detekce směru pohledu a mrkání jsem se rozhodl implementovat do software simulátoru knihovny OpenFace z následujících důvodů:

1. Je to otestovaný přístup, který splňuje předpoklady pro využití na simulátoru, chyba odhadu směru pohledu se pohybuje okolo 10° což odpovídá průměrné chybě určení POG 10-15 cm
2. Je naprogramovaný v C++ - je kompatibilní s Unreal Engine
3. Kromě nutných funkcí pro detekci mrkání a směr pohledu obsahuje i funkce pro detailní analýzu všech možných výrazů (Action Units), které lze v dalším vývoji využít pro přesnější odhad chování a únavy řidiče
4. Obsahuje robustní landmark detektor, který spolehlivě určuje významné body obličeje i při velkých úhlech natočení hlavy

4 Implementace do software simulátoru

Finálním a nejsložitějším krokem mojí práce bylo implementovat algoritmus do software simulátoru. Bylo nutné se nejprve naučit pracovat v Unreal Engine, což je jak již bylo řečeno programovací prostředí pro tvorbu her. Cílem bylo, aby do výstupního csv souboru přibyly informace o směru pohledu a mrkání.

4.1 Popis existujícího software

Hotový software je ve formě hry, spustitelné pomocí souboru SimulatorSECREDAS.exe. Při spuštění simulátoru se nejdříve načte úvodní obrazovka s menu, kde je možné si vybrat z několika různých testů (měření). První variantou je freeride course. Tato úroveň je zde, aby si uživatel zvykl na ovládání auta, volant a pedály. V této úrovni není žádné měření.

Druhá varianta je měření step response - odezvy na požadavek změny směru. Při výběru této úrovně je nejprve uživatel vyzván k zadání cesty k výstupnímu souboru, do které se zapisují následující údaje:

Time(ms)	X pos	Y pos	SplineDistance	Velocity	WheelAngle	HeartBeat_ms
...	...	...	...	...	...	...

Tab. 4.1: Formát výstupního souboru

- Čas od spuštění v ms
- Souřadnice auta ve scéně X pos, Y pos
- Vzdálenost od požadované trajektorie auta SplineDistance
- Rychlost auta Velocity
- Úhel natočení kol WheelAngle
- HeartBeat_ms pro záznam tepové frekvence uživatele

Třetí variantou je keep velocity measurement (měření udržování rychlosti). Uživatel nejprve zadá cestu k výstupnímu souboru a poté je měření spuštěno. Po nastaveném čase je umístěna do cesty srnka a zjišťuje se, jak rychle je uživatel schopen zareagovat. Ve výstupním souboru je navíc bool proměnná MannequinSpawn, která signalizuje, že se na cestě srnka nachází.

Poslední varianta je highway measurement. Tato úroveň byla vytvořena pro unavení řidiče. Uživatel jede v autě po dálnici, která je rovná nebo se mírně zatáčí.

4.1.1 Princip funkce úrovně

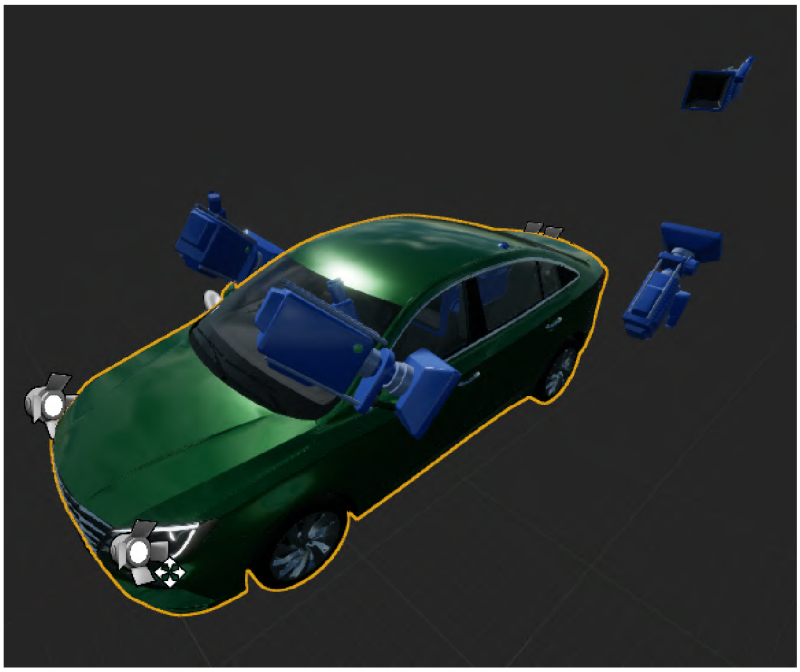
Každá úroveň využívá v základu 3 důležité typy blueprintů:

1. GameMode Blueprint
2. Level Blueprint
3. Blueprint class

Základem pro každou úroveň je GameMode blueprint cds_gameinstance. Obsahuje globální proměnné, které určují chování simulátoru. Globální proměnné lze poté nastavovat a číst v dalších blueprintech pomocí reference na cds_gameinstance a převodu pomocí funkce cast to cds_gameinstance. Dále je důležitý GameMode blueprint cds_mode_highway_step_response, který je nastavený jako override GameMode blueprint. Zde se programuje logika hry - jak se hráč připojí do úrovně, kdy se uživateli zobrazí pokyny změny směru, za jakých podmínek se úroveň pozastaví či ukončí.

Level Blueprint je vytvořen pro každou úroveň automaticky. Je to blueprint, ve kterém lze nastavovat scénáře specifické pro každou úroveň. V případě úrovně Keep Velocity Measurement je v Level Blueprintu zajištěno přidání zvěře na vozovku po nastaveném čase.

Blueprint Class je blueprint, který umožňuje vytvářet objekty ve scéně a přidat jim funkcionalitu. V případě simulátoru je typickým příkladem třída sedan_car, což zahrnuje tvar, texturu auta, kamery ve scéně, chování auta při zatáčení, dále řeší zrychlování auta při přidání plynu a podobně. Na obrázku 4.1 je prostředí, ve kterém je možné přidávat kamery k autu, měnit jeho barvu, světla atd.



Obr. 4.1: Grafický editor Blueprint Class sedan_car

4.2 Popis úpravy software

Zde je výčet všech kroků, jak bylo potřeba software upravit. Následující body dále rozepíšu.

- Přidání knihoven třetích stran tak, aby je bylo možné využívat v C++ kódu Unreal Engine
- Napsání kódu, který bude zajišťovat snímání uživatele webkamerou a analýzu snímků
- Zamknutí pohledu kamery tak, aby jej nebylo možné měnit myší
- Přidat zapisování dat do csv souboru

4.2.1 Přidání knihoven do UE

Knihovna OpenFace je napsaná v C++. Skládá se z knihoven FaceAnalyser, GazeAnalyzer a LandmarkDetector. Pro správnou funkci těchto knihoven jsou dále potřeba knihovny OpenCV (algoritmy počítačového vidění), dlib (strojové učení), OpenBLAS (optimalizované počítání s vektory a maticemi). Všechny tyto knihovny je nutné správně linkovat. OpenCV a OpenBLAS jsou dynamicky linkované knihovny, ostatní jsou linkované staticky. Přidání knihoven je možné realizovat v souboru SimulatorSECREDAS.Build.cs pomocí funkcí:

- Cesta k hlavičkovým souborům knihovny - PublicIncludePaths.AddRange();
- Cesta k .lib souborům - PublicLibraryPaths.Add();
- Cesta ke konkrétnímu .lib souboru knihovny (např. dlib.lib) - PublicAdditionalLibraries.Add();
- Přidání dll - PublicDelayLoadDLLs.Add();
- Přidání závislostí - RuntimeDependencies.Add();

Problémy s linkováním knihoven

Správné sestavení projektu tak, aby všechny knihovny fungovaly správně, byl zpočátku neskutečně složitý úkol. Není mnoho návodů, jak správně knihovny přidávat do UE, takže když něco nefungovalo, bylo těžké zjistit proč. Nejprve jsem podle návodu [6] zjistil, jak se správně přidává knihovna OpenCV, ostatní knihovny jsem pak přidával analogicky. Některé funkce knihoven však měly stejný název jako funkce Unreal Engine, při kompilaci pak docházelo ke kolizi. Toto se nejvíce týkalo knihoven OpenCV a dlib, kde jsem musel měnit názvy funkcí a proměnných tak, aby ke kolizím nedocházelo (před název dlib funkce jsem přidával d_). Nakonec se po dlouhém snažení vše podařilo a bylo možné využívat knihovnu Openface.

4.2.2 Analýza pohledu a mrkání

Všechny metody spojené se sledováním a analýzou obličeje uživatele jsem sjednotil do třídy UserBehaviorTracking. Třída dědí UE třídu ActorComponent - využívám zde metodu BeginPlay(). Metody jsou vždy označeny makrem UFUNCTION(), aby je bylo možné využívat jako blueprint funkce.

BeginPlay()

Tato metoda je volána pokaždé, když je třída jakkoli umístěna do hry. K autu (sedan_car) jsem přidal instanci třídy UserBehaviorTracking. Ve třídě UserBehaviorTracking využívám metodu BeginPlay() pro vytvoření instance třídy FaceAnalyser, dále pro inicializaci LandmarkDetectoru a také otevření webkamery pro čtení snímků. Metoda BeginPlay() není použita v blueprintu.

compute()

Metoda compute() je volána při každém obnovení scény. Metoda je spouštěna v každém blueprint levelu pomocí impulzu od Tick funkce levelu. Pomocí funkce LandmarkDetector::DetectLandmarksInVideo jsou nejprve detekovány významné body obličeje. V dalším kroku se použije GazeAnalysis::EstimateGaze a poté GazeAnalysis::GetGazeAngle. Výsledkem je vektor obsahující úhel α (horizontální úhel pohledu v radiánech) a β (vertikální úhel pohledu v radiánech). Dále je pomocí funkce LandmarkDetector::GetPose určena pozice hlavy v souřadnicovém systému, jehož počátkem je střed kamery, osa Z je rovnoběžná s osou kamery, x je horizontální osa, y vertikální. Souřadnicový systém je na obrázku 4.4

4.2.3 Zamknutí pohledu kamery

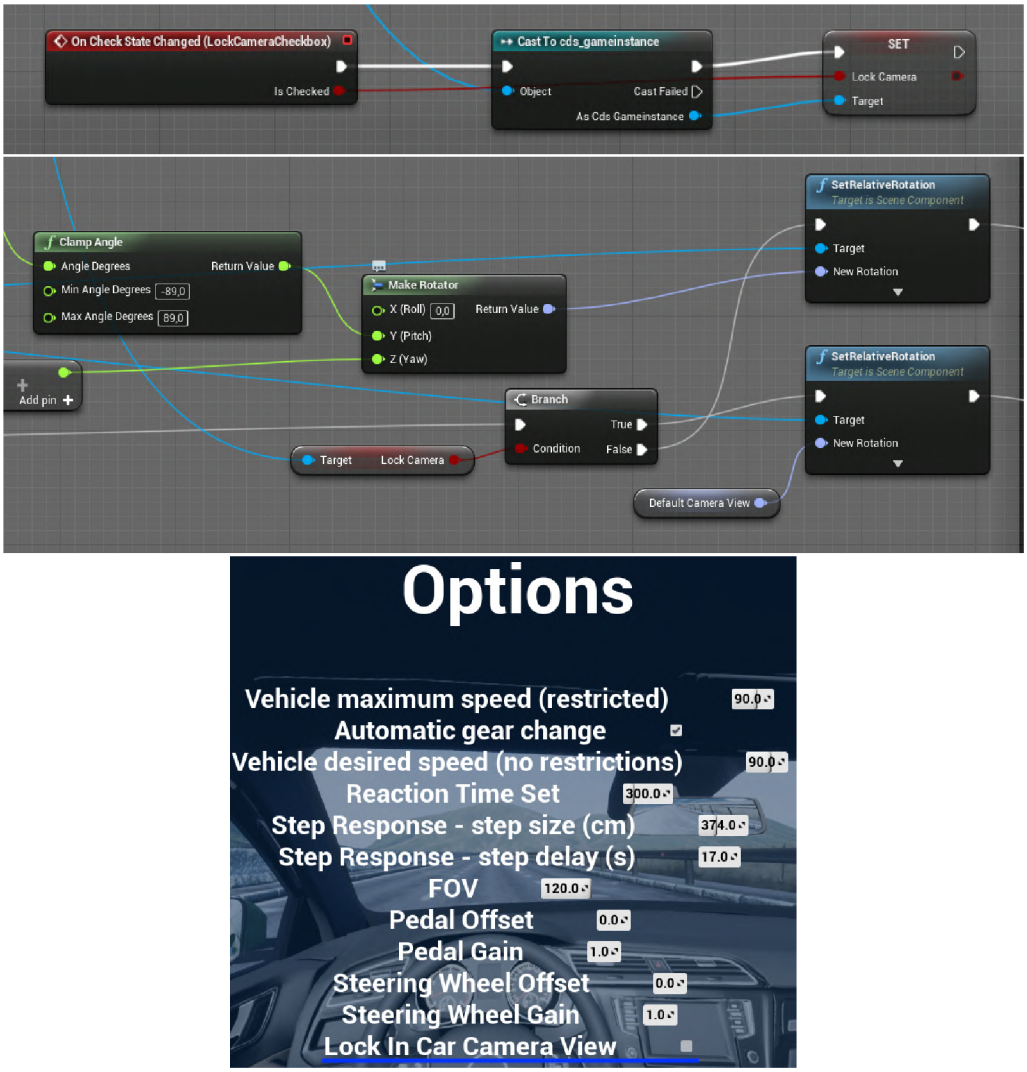
Bylo potřeba nastavit pohled tak, aby jej nebylo možné měnit myší. Ve výchozím stavu je možné se rozhlížet pohybem myši v rámci celého interiéru vozidla. Pokud však máme zpětně analyzovat místo pohledu uživatele, musí na obrazovce být vždy stejný obraz. Proto jsem do proměnných v cds_gameinstance přidal bool LockCamera. Pokud je hodnota true, bude nastaven výchozí pohled v automobilu a nebude jej možné pomocí myši měnit. Hra obsahuje dvě nastavení, do jednoho se dostaneme, když pozastavíme hru, do druhého hned při spuštění hry z hlavního menu. Do obou jsem přidal checkbox, jehož hodnota je navázána na bool proměnou v blueprintu. Výchozí pohled jsem nastavil pomocí konstantního vektoru [yaw, pitch, roll] tak, aby ve výhledu byla vidět obě zpětná zrcátka.

Na obrázku 4.2 jsou tři snímky obrazovky, jsou zde vidět konkrétní úpravy. První snímek ukazuje nastavení globální proměnné funkcí OnCheckStateChanged.

Při kliknutí na checkbox Lock Camera se tato funkce zavolá a nastaví globální proměnnou LockCamera.

Druhý snímek zobrazuje část blueprint třídy sedan_car zodpovědnou za změnu pohledu myší. Zde jsem přidal funkci Branch, jejíž podmínkou je proměnná LockCamera. Pokud je LockCamera true, nastaví se funkcí SetRelativePosition výchozí pohled a nelze ho měnit.

Třetí snímek ukazuje upravené menu.



Obr. 4.2: Zamknutí kamery

4.2.4 Zapisování dat do csv

Data se zapisují pomocí funkce LogDataToFileCoord či LogDataToFileMannequin (v úrovni Keep Velocity measurement) do csv souboru. Pro tento účel jsem přidal globální proměnné do cds_gameinstance:

- směr pohledu - float GazeDirectionX (horizontální úhel pohledu) a GazeDirectionY (vertikální úhel pohledu)
- souřadnice hlavy - float HeadPositionX, HeadPositionY, HeadPositionZ
- mrkání - float BlinkEstimation (hodnota určena pomocí OpenFace AU45 v rozsahu 0-5)

Vybral jsem tyto informace, protože jsou nejvíce univerzální, a v rámci post-processingu z nich lze extrahovat místo pohledu na obrazovce a frekvence a délka mrkání.

Tyto proměnné aktualizuji v sedan_car. Na prvním snímku obrázku 4.3 je princip zápisu pozice hlavy X pomocí metody GetHeadPositionX třídy UserBehaviorTracking do globální proměnné HeadPositionX. Funkce GetHeadPositionX je spouštěna pomocí EventTick, který je volán při každém obnovení scény simulátoru.

Z globálních proměnných jsou pak tyto informace zapisovány pomocí funkce LogDataToFileCoord využité v GameMode override blueprintu každého levelu. Funkci jsem upravil tak, aby obsahovala další vstupní proměnné a zapisovala je v csv souboru do dalších sloupců. Obrázek 4.3 je spíše ilustrační, pro přehlednost ukazuje zápis pouze jedné proměnné.

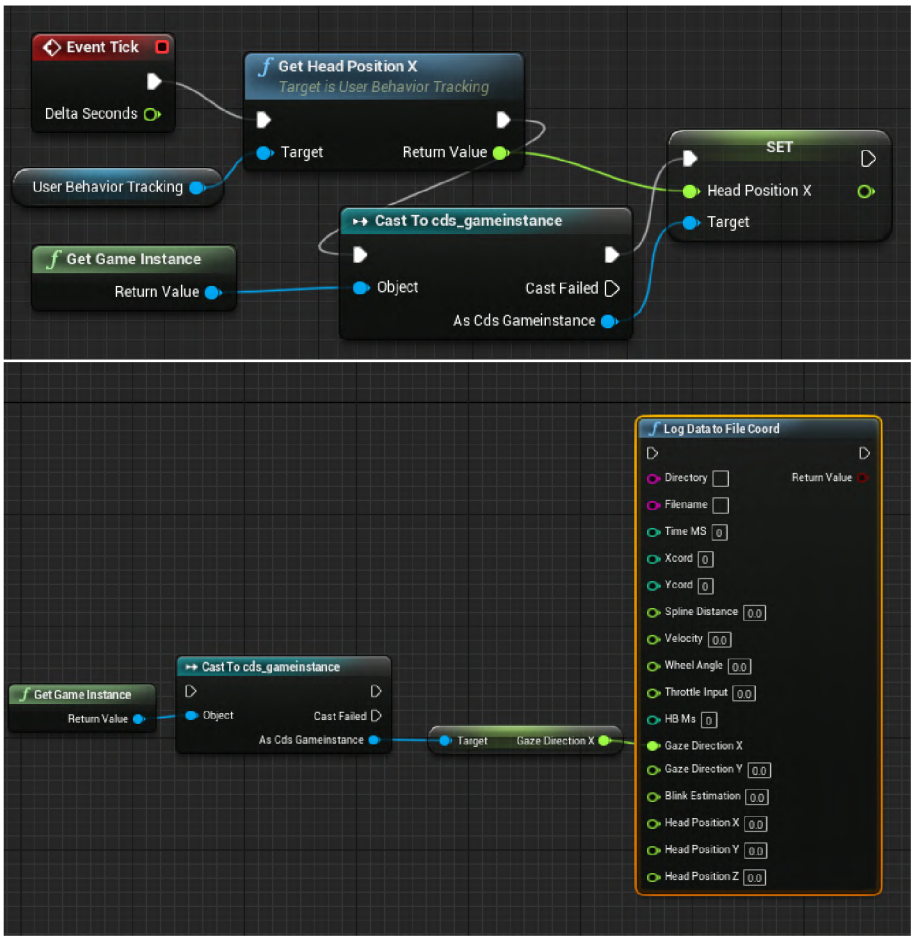
4.3 Analýza dat z csv souboru

Posledním krokem je analýza vytvořených dat. Dosud to bylo realizováno v Matlabu, proto jsem se rozhodl dodělat skript analyzedata.m, který z úhlů pohledu a pozice hlavy vypočítá průsečík pohledu s monitorem a zobrazí jej v grafu. Dále z csv souboru spočítám frekvenci a průměrnou délku mrkání v rámci celého měření.

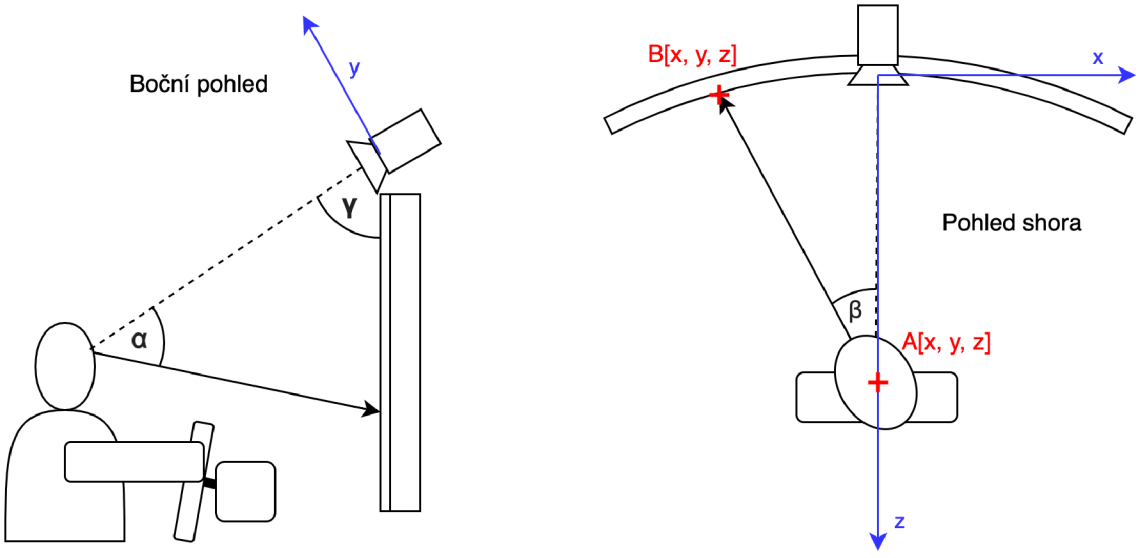
Model scény je na obrázku 4.4. Zde je vidět souřadnicový systém, se kterým pracujeme a důležité parametry.

Je vidět, že souřadnicový systém (osy označeny modře) určuje kamera. Úhel γ určuje natočení kamery vůči monitoru. Pro výpočet je nutné znát bod A, který označuje pozici hlavy v prostoru, dále rozměry monitoru a vzdálenost monitoru od kamery. Úhly α a β určují směr pohledu.

V Matlabu je nejdříve načten vstupní csv soubor pomocí funkce *readtable*(). Ze souboru jsou vybrány sloupce s úhly směru pohledu, souřadnice hlavy a odhad mrkání.



Obr. 4.3: Zápis do csv



Obr. 4.4: Důležité parametry scény pro postprocessing

Směr pohledu

Úhly směru pohledu jsou pomocí trigonometrických funkcí sin a cos přepočítány na jednotkový směrový vektor.

$$\begin{aligned} x &= sin(\alpha) * cos(\beta) \\ y &= sin(\beta) \\ z &= cos(\alpha) * cos(\beta) \end{aligned} \tag{4.1}$$

Dále je vytvořen 3D rastr XY s rozměry monitoru pomocí funkce meshgrid. Souřadnice Z jsou vypočítány pomocí rovnice kružnice o poloměru 1800 mm (zakřivení monitoru).

$$\begin{aligned} &a = 1,\ b = -3200,\ c = x^2 \\ &D = b^2 - 4 * a * c \\ &z = (-b - sqrt(D))/2/a + cam\_offset \end{aligned} \tag{4.2}$$

Celá plocha je poté rotována okolo osy x o úhel γ. Pro čitelnost je na tuto plochu namapován snímek obrazovky simulátoru. Nakonec je uživateli zobrazena přímka protínající obrazovku simulátoru.

Obr. 4.5: Grafické znázornění směru pohledu

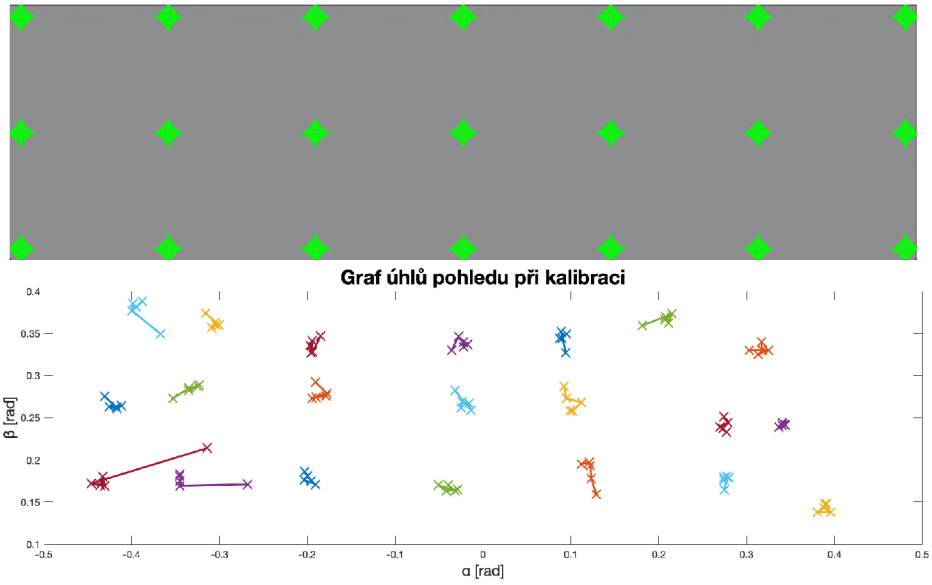
4.4 Kalibrace a testování měření směru pohledu

Geometrický přístup k výpočtu místa pohledu na obrazovce není moc přesný, protože úhly pohledu z OpenFace jsou většinou menší než reálné. Sám autor OpenFace T. Baltrusaitis tento problém zodpovídal na stránkách repozitáře. Doporučuje, aby každý uživatel provedl kalibraci před měřením.

Proto jsem vytvořil v Pythonu kalibrační program (main.py v elektronické příloze), který při spuštění uživateli zobrazí okno na celou obrazovku monitoru a ukazuje body v rámci celé obrazovky monitoru a vytváří snímky uživatele. Snímky jsou

ukládány do dvou složek, imgs_calibration pro kalibrační snímky a imgs_validation pro testovací snímky.

Celkem je na monitoru rozmístěno 21 bodů v rastru 7x3, tomu odpovídá rozteč 27x18 cm. Každý bod je zobrazen, poté se čeká 1.5 vteřiny, aby se uživatel stačil podívat na nově zobrazený bod a vytvoří se 10 snímků frekvencí 10 Hz. Pořadí bodů je zvoleno náhodně.



Obr. 4.6: Rozložení 21 kalibračních bodů na obrazovce a odpovídající záznam z OpenFace

Dále je pomocí modulu subprocess zavolán program FeatureExtraction.exe, který zajistí analýzu kalibračních snímků pomocí OpenFace knihovny. Výstupem je csv soubor, kde každý snímek odpovídá jednomu řádku, který obsahuje výsledek analýzy. Každý řádek obsahuje 1428 údajů, patří sem číslo snímku, směr pohledu, souřadnice 2D i 3D významných bodů, pozice hlavy a natočení hlavy, hodnoty AU(mrkání, úsměv, zvednutí obočí). Data jsou z něj načtena a je vytvořena matice X [α, β, head_pose_X, head_pose_Y, head_pose_Z, yaw, pitch, roll] a dále je vytvořena matice souřadnic bodů zobrazených na monitoru v px ve správném pořadí y [x, y].

Kalibrace je provedena jako natrénování machine learning modelu. Otestoval jsem následující přístupy:

1. Lineární regrese
2. Support Vector regrese (SVR) s lineárním kernelem

Oba dva přístupy jsou lineární, postupným testováním jsem zjistil, že tyto přístupy jsou nejúspěšnější v regresi nových dat.

Kalibrace pomocí lineární regrese byla řešena pomocí modulu SciKit Learn LinearRegression. Tato funkce vytvoří lineární model:

$$y = w_1 + w_2 X_1 + w3 X_2 + \ldots + w_n X_n \tag{4.3}$$

Kde koeficienty $w = w_1, w_2, ..., w_n$ jsou spočteny tak, aby se minimalizovala odchylka kvadrátu.

4.4.1 Dataset

Pro určení přesnosti OpenFace jsem vytvořil dataset na simulátoru řízení. Dataset tvořilo 10 osob mužského pohlaví. Jeden subjekt měl brýle, ostatní bez brýlí. Každý nahrál celkem 210 snímků sledováním všech náhodně zobrazených bodů. Každý bod tedy obsahuje 10 snímků od každé osoby.

Příklady několika snímků z datasetu jsou na obrázku 4.7

Obr. 4.7: Dataset pořízený na simulátoru řízení

4.4.2 Výsledky

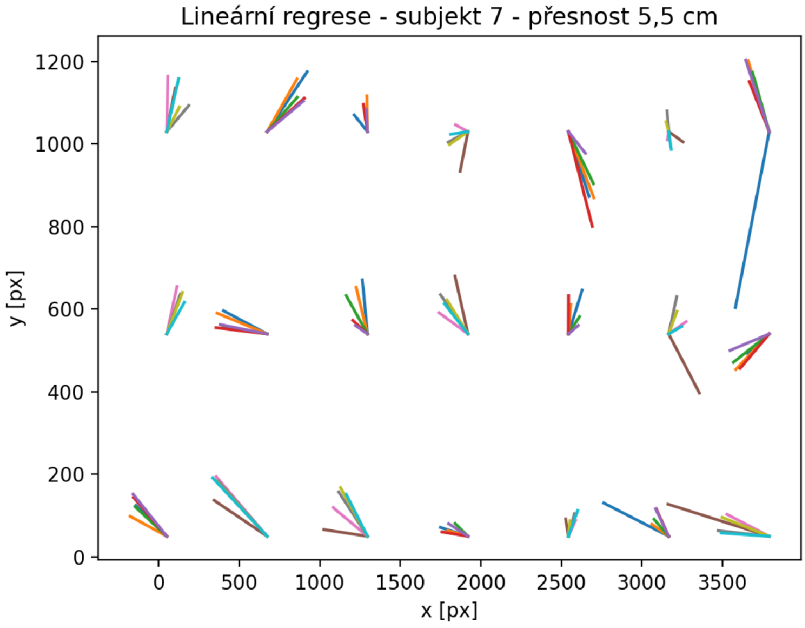
V tabulce 4.2 jsou výsledky měření směru pohledu pomocí OpenFace. Modely byly natrénovány na 105 snímcích od každého uživatele, testování proběhlo pomocí validační části datasetu. Vyzkoušel jsem několik variant. Ukázalo se, že nejlepší výsledky jsou pomocí lineární regrese s použitím všech vstupních dat včetně natočení hlavy a při použití kalibrace. Kalibrace však není nezbytně nutná. Protože jsem do software simulátoru implementoval zapisování pouze úhlů pohledu a souřadnic hlavy, jako

nejlepší varianta se jeví lineární regrese ze všech dat bez kalibrace. Chyba je zde průměrně 10,4 cm. SVR se mi nepodařilo nastavit tak, aby fungovala lépe. Zkoušel jsem i regresi SVR s jinými kernely, ale výsledky byly jenom horší.

První řádek tabulky je výsledek geometrického přepočtu směru pohledu na místo pohledu na obrazovce. K tomu byl použit skript gazecalculation.m.

Model	Kalibrace	Vstupní data	Přesnost [cm]
Výpočet	ne	(α, β, x, y, z)	17,3
Lineární regrese	ano	(α, β, x, y, z)	11,1
Lineární regrese	ano	$(\alpha, \beta, x, y, z, \phi, \theta, \psi)$	9,9
Lineární regrese	ne	(α, β, x, y, z)	11,7
Lineární regrese	ne	$(\alpha, \beta, x, y, z, \phi, \theta, \psi)$	10,4
SVR	ano	$(\alpha, \beta, x, y, z, \phi, \theta, \psi)$	16,2

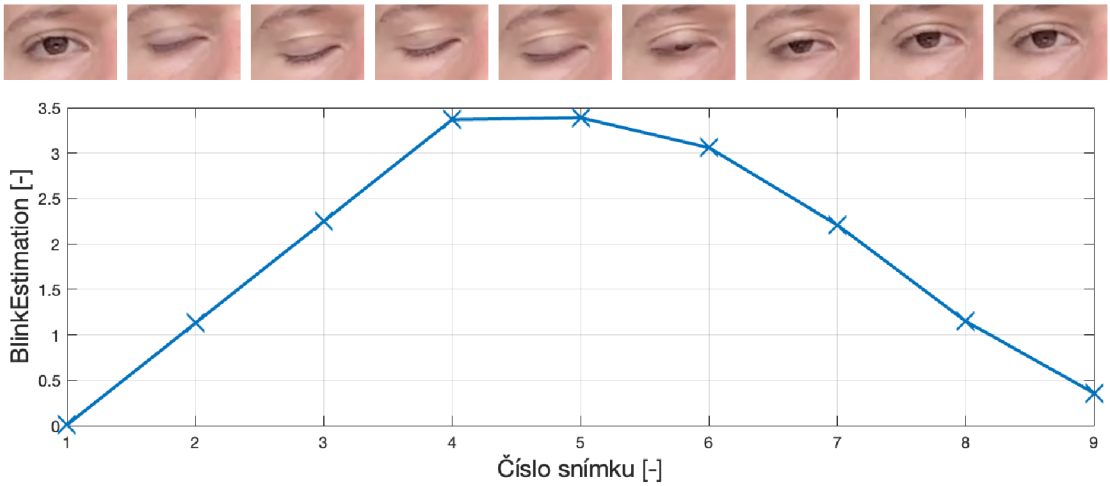
Tab. 4.2: Výsledky měření (úhly pohledu: α, β, pozice hlavy: x, y, z, natočení hlavy: ϕ, θ, ψ)



Obr. 4.8: Ukázka detekce směru pohledu - Lineární regresor natrénován na celém datasetu, testováno na validační části uživatele 7

4.5 Testování detekce mrkání

Pro testování detekce mrkání jsem nahrál video a z něj extrahoval snímky na kterých mrkám. Příklad těchto snímků je na obrázku 4.9.

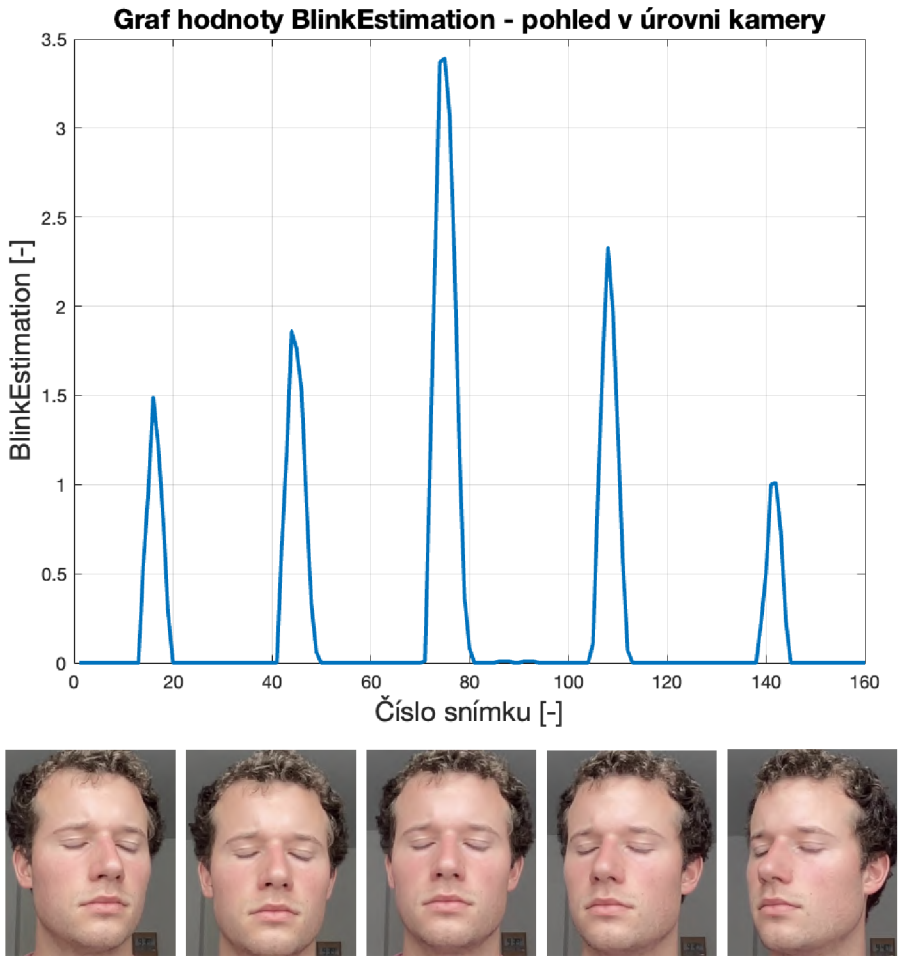


Obr. 4.9: Ukázka detekce mrkání (snímky odpovídají hodnotám v grafu

Hodnoty, které OpenFace generuje pro mrkání, jsou v rozsahu 0-5, kde 0 znamená oko otevřené, 5 oko zavřené. Při pohledu dolů se detekce mrkání zhoršuje, protože oko se mírně přivře a hodnota narůstá. V analyzovaném videu jsem se nejprve díval v úrovni kamery 4.10, poté mírně dolů 4.11. Při porovnání grafů je zřejmé, že detekce je více spolehlivá v prvním případě. Z grafu je možné jednoznačně určit mrknutí. Při pohledu dolů 4.11 je hodnota BlinkEstimation nenulová mezi každým mrknutím. Dalším jevem, který můžeme pozorovat, je závislost detekce mrknutí na natočení hlavy. Čím více je hlava natočená, tím hůře se mrknutí detekuje. Faktem však zůstává, že většinu času uživatel simulátoru sleduje vozovku, která je v oblasti spolehlivé detekce mrkání.

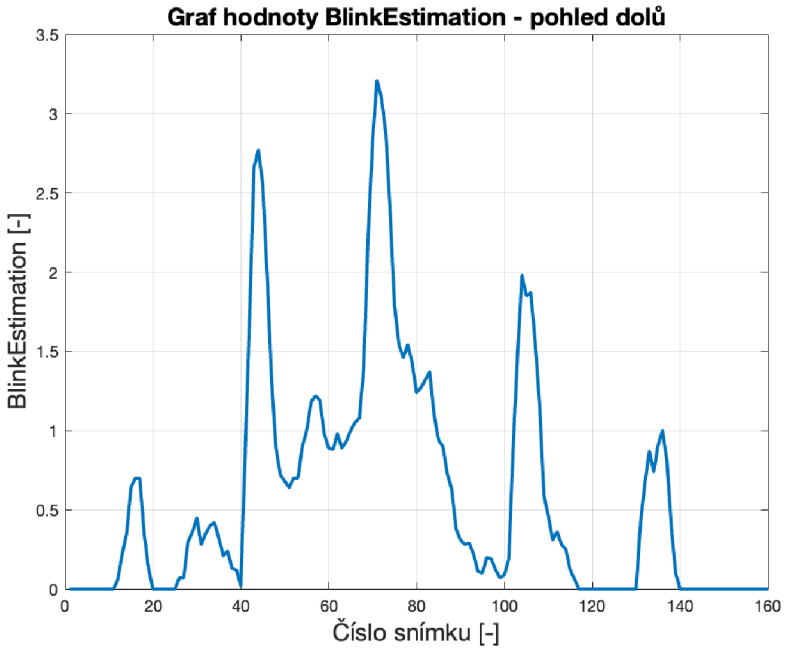
4.5.1 Výpočet frekvence a délky mrkání

Frekvence a délka mrkání je spočtena z csv souboru pomocí Matlab skriptu analyzedata.m. Frekvenci mrkání lze vyčíslit jako počet mrknutí uživatele za měření děleno celkový čas měření. Počet mrknutí uživatele spočítám tak, že zjistím, kolikrát byla hodnota BlinkEstimation větší než hranice (hranici jsem podle grafů nastavil na 1.5, abych co nejvíce omezil falešně detekovaná mrknutí při pohledu dolů). Pro tento účel používám funkci bwlabel(blinkestimation>1.5). Celkový čas měření se zapisuje do prvního sloupce csv souboru. V posledním řádku prvního sloupce je celkový čas měření v milisekundách. Stačí převést na minuty a máme požadovanou hodnotu frekvence mrkání za minutu.



Obr. 4.10: Graf BlinkEstimation - pohled v úrovni kamery. Každý snímek odpovídá špičce v grafu.

V Matlabu jsem pro délku mrkání použil druhý výstup funkce bwlabel. Tento výstup obsahuje matici, která je rozměry identická se vstupní, ale každé mrknutí je zde očíslováno (pokud je vstup [0 0 1 0 0 1 1 1 0 1 1 0], výstup bude [0 0 1 0 0 2 2 2 0 3 3 0]). Takto lze jednotlivá mrknutí rozlišit a zjistit jejich délku (využít matici pro indexaci časového sloupce). Nakonec se spočítá průměrná délka všech mrknutí.



Obr. 4.11: Graf BlinkEstimation - pohled dolů

Závěr

V rámci diplomové práce byla provedena rešerše existujících přístupů pro detekci mrkání a odhadu směru pohledu ze snímků obličeje. Některé přístupy byly vyzkoušeny s pomocí webkamery v domácích podmínkách, některé na simulátoru řízení.

První způsob, který jsem vyzkoušel, byla detekce frekvence a délky mrkání pomocí výpočtu EAR. S pomocí OpenCV knihovny byl vytvořen algoritmus, který určil významné body obličeje a s jejich pomocí i eye aspect ratio. Algoritmus byl otestován s pomocí webkamery a bylo zjištěno, že dokáže mrknutí detekovat.

Dále byl otestován algoritmus pro určení natočení hlavy. 2D významné body obličeje byly pomocí Pose Estimation funkce spojeny s 3D body modelu hlavy. Výstupem funkce je vektor natočení hlavy.

Nejprve jsem předpokládal, že pro detekci směru pohledu natrénuji konvoluční neuronovou síť. Tento přístup byl inspirován [12], [21] a dalšími články. Pro trénink byl vytvořen dataset přibližně 10 tisíc snímků. Podařilo se mi síť natrénovat tak, aby směr pohledu klasifikovala do zón, ale nebyl jsem si jistý, zda bude tento přístup robustní a funkční i pro další osoby.

Na simulátoru byla vyzkoušena knihovna OpenFace 2.0. Zjistil jsem, že dokáže určit směr pohledu a zároveň i mrkání. Dokonce umožňuje sledovat i další výrazy tváře, což se může hodit pro analýzu nálady či ospalosti řidiče. Rozhodl jsem se, že tuto knihovnu implementuji do software simulátoru.

Software simulátoru byl vytvořen v Unreal Engine. Do software byly přidány knihovny OpenFace, OpenCV, dlib a OpenBLAS. Byly implementovány funkce, které při spuštění simulátoru zaznamenávají snímky uživatele a pomocí OpenFace je analyzují a výsledky zapisují do csv souboru. Další úprava software spočívala v zamčení pohledu kamery ve vozidle, aby uživatel nemohl myší s pohledem otáčet.

Pro otestování finálního řešení jsem s pomocí kolegů na VUT vytvořil dataset. Dataset se tvořil sledováním bodu, který se zobrazoval na různých místech monitoru v mřížce. Celkem bylo nahráno 2100 snímků 10 osob. Dataset byl využit pro otestování OpenFace knihovny. Zjistil jsem, že přepočet úhlů směru pohledu a pozice hlavy v 3D prostoru na místo pohledu na monitoru není přesný. S pomocí datasetu jsem vytvořil linární regresor, který dokázal přesnost určení pohledu na obrazovce vylepšit až na průměrných 9.9 cm při zkalibrování uživatelem. Nová data je však možné analyzovat i bez kalibrace při průměrné přesnosti 10.4 cm.

OpenFace knihovna byla dále využita i pro detekci mrkání. Výpočet frekvence a délky mrkání byl realizován z csv dat v software Matlab. Funkčnost detekce mrkání byla ověřena na testovacím videu. Nejlépe detekce funguje v případě, kdy se člověk dívá na vozovku. V případě pohledu dolů či do stran se detekce mírně zhoršuje.

Výsledkem práce je tedy upravený simulátor řízení, který nově dokáže určovat

směr pohledu a mrkání uživatele v průběhu řízení. Tyto informace lze využít k bližšímu pochopení chování řidiče.

Literatura

[1] BALTRUSAITIS, T.; ROBINSON, P.; MORENCY, L.-P.: Constrained Local Neural Fields for Robust Facial Landmark Detection in the Wild. 12 2013, doi:10.1109/ICCVW.2013.54.

[2] BALTRUSAITIS, T.; ZADEH, A.; LIM, Y. C.; aj.: OpenFace 2.0: Facial Behavior Analysis Toolkit. In *2018 13th IEEE International Conference on Automatic Face Gesture Recognition (FG 2018)*, 2018, s. 59–66, doi:10.1109/FG.2018.00019.

[3] BEN-HUR, A.; WESTON, J.: A user's guide to support vector machines. *Methods in molecular biology*, ročník 609, 2010: s. 223–39.

[4] DALAL, N.; TRIGGS, B.: Histograms of oriented gradients for human detection. In *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, ročník 1, 2005, s. 886–893 vol. 1, doi:10.1109/CVPR.2005.177.

[5] FUNES MORA, K.; MONAY, F.; ODOBEZ, J.-M.: EYEDIAP: a database for the development and evaluation of gaze estimation algorithms from RGB and RGB-D cameras. 03 2014, s. 255–258, doi:10.1145/2578153.2578190.

[6] GINKU: Integrating OpenCV into Unreal Engine 4 [online]. Poslední aktualizace 2020 [cit. 15. 2. 2022].
URL `https://unreal.gg-labs.com/wiki-archives/ar-vr/`

[7] HAREZLAK, K.; KASPROWSKI, P.; STASCH, M.: Towards Accurate Eye Tracker Calibration – Methods and Procedures. *Procedia Computer Science*, ročník 35, 12 2014, doi:10.1016/j.procs.2014.08.194.

[8] KAZEMI, V.; SULLIVAN, J.: One Millisecond Face Alignment with an Ensemble of Regression Trees. 06 2014, doi:10.13140/2.1.1212.2243.

[9] KHABARLAK, K.; KORIASHKINA, L.: Fast Facial Landmark Detection and Applications: A Survey. 04 2022, doi:10.13140/RG.2.2.32735.07847/1.

[10] KHAN, M. Q.; LEE, S.: Gaze and Eye Tracking: Techniques and Applications in ADAS. *Sensors*, ročník 19, č. 24, Dec 2019: str. 5540, ISSN 1424-8220, doi:10.3390/s19245540.

[11] KIM, W.; JUNG, W.-S.; CHOI, H. K.: Lightweight Driver Monitoring System Based on Multi-Task Mobilenets. *Sensors*, ročník 19, č. 14, 2019, ISSN 1424-8220, doi:10.3390/s19143200.

[12] KRAFKA, K.; KHOSLA, A.; KELLNHOFER, P.; aj.: Eye Tracking for Everyone. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, s. 2176–2184, doi:10.1109/CVPR.2016.239.

[13] LE, V.; BRANDT, J.; LIN, Z.; aj.: Interactive Facial Feature Localization. In *Computer Vision – ECCV 2012*, editace A. FITZGIBBON; S. LAZEBNIK; P. PERONA; Y. SATO; C. SCHMID, Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, ISBN 978-3-642-33712-3, s. 679–692.

[14] MATAS, J.: Detekce objektu pomocí scanning window [online]. [cit. 24. 1. 2022]. URL `http://vision.uamt.feec.vutbr.cz/ROZ/lectures/`

[15] PARK, S.; ZHANG, X.; BULLING, A.; aj.: Learning to Find Eye Region Landmarks for Remote Gaze Estimation in Unconstrained Settings. In *Proceedings of the 2018 ACM Symposium on Eye Tracking Research & Applications*, ETRA '18, New York, NY, USA: Association for Computing Machinery, 2018, ISBN 9781450357067, doi:10.1145/3204493.3204545.

[16] SAGONAS, C.; ANTONAKOS, E.; TZIMIROPOULOS, G.; aj.: 300 Faces In-The-Wild Challenge: database and results. *Image and Vision Computing*, ročník 47, 2016: s. 3–18, ISSN 0262-8856, doi:https://doi.org/10.1016/j.imavis.2016.01.002, 300-W, the First Automatic Facial Landmark Detection in-the-Wild Challenge.

[17] SMOLA, A. J.; SCHÖLKOPF, B.: A tutorial on support vector regression. *Statistics and computing*, ročník 14, č. 3, 2004: s. 199–222, doi:https://doi.org/10.1023/B:STCO.0000035301.49549.88.

[18] SUN, Y.; WANG, X.; TANG, X.: Deep Convolutional Network Cascade for Facial Point Detection. In *2013 IEEE Conference on Computer Vision and Pattern Recognition*, 2013, s. 3476–3483, doi:10.1109/CVPR.2013.446.

[19] TOBII: Eye tracker accuracy and precision [online]. [cit. 12. 4. 2022]. URL `https://www.tobiipro.com/learn-and-support/learn/`

[20] VIOLA, P.; JONES, M.: Rapid object detection using a boosted cascade of simple features. In *Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition. CVPR 2001*, ročník 1, 2001, s. I–I, doi:10.1109/CVPR.2001.990517.

[21] VORA, S.; RANGESH, A.; TRIVEDI, M.: Driver Gaze Zone Estimation Using Convolutional Neural Networks: A General Framework and Ablative Analysis.

IEEE Transactions on Intelligent Vehicles, ročník PP, 02 2018, doi:10.1109/TIV.2018.2843120.

[22] WOOD, E.; BALTRUAITIS, T.; ZHANG, X.; aj.: Rendering of Eyes for Eye-Shape Registration and Gaze Estimation. In *2015 IEEE International Conference on Computer Vision (ICCV)*, 2015, s. 3756–3764, doi:10.1109/ICCV.2015.428.

[23] WOOD, E.; BALTRUŠAITIS, T.; MORENCY, L.-P.; aj.: Learning an Appearance-Based Gaze Estimator from One Million Synthesised Images. In *Proceedings of the Ninth Biennial ACM Symposium on Eye Tracking Research & Applications*, 2016, s. 131–138.

[24] WOOD, E.; BALTRUSAITIS, T.; ZHANG, X.; aj.: Rendering of Eyes for Eye-Shape Registration and Gaze Estimation. 2015, doi:10.48550/ARXIV.1505.05916.

[25] YOO, D.; CHUNG, M.: A novel non-intrusive eye gaze estimation using cross-ratio under large head motion. *Computer Vision and Image Understanding*, ročník 98, 04 2005: s. 25–51, doi:10.1016/j.cviu.2004.07.011.

[26] ZAFEIRIOU, S.; CHRYSOS, G.; ROUSSOS, A. T.; aj.: The 3D Menpo Facial Landmark Tracking Challenge. 10 2017, s. 2503–2511, doi:10.1109/ICCVW.2017.16.

[27] ZHANG, X.; PARK, S.; BEELER, T.; aj.: ETH-XGaze: A Large Scale Dataset for Gaze Estimation under Extreme Head Pose and Gaze Variation. 2020, doi:10.48550/ARXIV.2007.15837.

[28] ZHANG, X.; SUGANO, Y.; FRITZ, M.; aj.: MPIIGaze: Real-World Dataset and Deep Appearance-Based Gaze Estimation. 2017, doi:10.48550/ARXIV.1711.09017.

[29] ZHOU, E.; FAN, H.; CAO, Z.; aj.: Extensive Facial Landmark Localization with Coarse-to-Fine Convolutional Network Cascade. In *2013 IEEE International Conference on Computer Vision Workshops*, 2013, s. 386–391, doi:10.1109/ICCVW.2013.58.

A Obsah elektronické přílohy

- `/` ..... kořenový adresář přiloženého archivu
  - `BlinkTest` ..... soubory pro testování detekce mrkání
    - `blinktest.m`
    - `blinktestfinal.avi`
    - `blinktestfinal.csv`
    - `blinktestfinal.mov`
  - `CNN` ..... Python projekt pro tvorbu CNN
    - `dataset` ..... Vytvořený dataset pro detekci zón pohledu
    - `haarcascades` ..... Haarovy kaskády pro detekci obličeje
    - `Model` ..... Natrénovaný model pro odhad zóny pohledu
    - `accuracyTest.py`
    - `createDataset.py`
    - `gazeEstimation.py`
    - `trainModel.py`
  - `HeadPose_EAR` ..... C++ projekt pro detekci mrkání a natočení hlavy
  - `OpenFace_test` .... Python projekt pro kalibraci a určení přesnosti detekce směru pohledu
    - `dataset` ..... Obsahuje kalibrační a validační snímky kolegů na VUT, včetně analýzy pomocí OpenFace a Matlab skriptu pro určení přesnosti OpenFace
    - `calibration.py`
    - `func.py`
    - `main.py`
    - `screenshot_simulator.png`
  - `analyzedata.m` ..... Skript pro výpočet průsečíku směru pohledu s monitorem a frekvence a délky mrkání
  - `Simulator_SECREDAS`
    - `cds_logdata_BP_function.h` ..... Upravená funkce zapisování dat do csv
    - `cds_logdata_BP_function.cpp`
    - `SECREDAS_simulator.Build.cs` . Zde jsem provedl linkování knihoven pro UE
    - `UserBehaviorTracking.h` ..... Třída pro analýzu uživatele kamerou
    - `UserBehaviorTracking.cpp`

Pro spuštění Python skriptů je třeba mít nainstalované knihovny:

- OpenCV
- Tensorflow 2.0
- NumPy
- Random
- Re
- Matplotlib
- Scikit-image