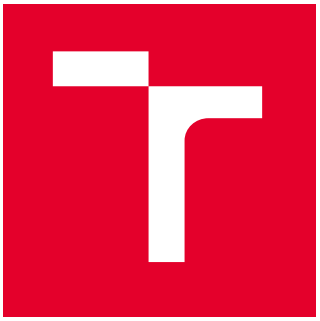


VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

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

STABILIZACE VIDEA NA RASPBERRY PI

VIDEO STABILIZATION ON RASPBERRY PI

BAKALÁŘSKÁ PRÁCE

BACHELOR'S THESIS

AUTOR PRÁCE — Alikhan Nurkhat

AUTHOR

VEDOUCÍ PRÁCE — Ing. Dominik Řičánek

SUPERVISOR

BRNO 2025

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

Semestrální práce

bakalářský studijní program **Automatizační a měřicí technika**

Ústav automatizace a měřicí techniky

Student: Alikhan Nurkhat ***ID:*** 242251

Ročník: 3 ***Akademický rok:*** 2024/25

NÁZEV TÉMATU:

Stabilizace videa na Raspberry Pi

POKYNY PRO VYPRACOVÁNÍ:

Úkolem studenta bude vytvořit algoritmus pro stabilizaci videa běžící v libovolném jazyce na platformě Raspberry Pi 4B. Tento algoritmus bude schopen odstranit vlivy malých otřesů kamery, a to co nejblíže běhu v reálném čase.

1. Proveďte rešerši exitujících algoritmů pro stabilizaci videa (např. využívajících optický tok).
2. Napište vybraný algoritmus ve vámi zvoleném jazyce s ohledem na danou platformu.
3. Otestujte algoritmus natočením videí na Raspberry Pi a porovnáním s nestabilizovaným videem. Pro úspěšné splnění semestrální práce je vyžadováno splnění bodu 1 zadání.

DOPORUČENÁ LITERATURA:

M. Grundmann, V. Kwatra and I. Essa, "Auto-directed video stabilization with robust L1 optimal camera paths," CVPR 2011, Colorado Springs, CO, USA, 2011, pp. 225-232, doi: 10.1109/CVPR.2011.5995525

Termín zadání: 16.9.2024 ***Termín odevzdání:*** 7.1.2025

Vedoucí práce: Ing. Dominik Řičánek

Ing. Miroslav Jirgl, Ph.D.
předseda rady studijního programu

ABSTRAKT

Tato bakalářská práce se zaměřuje na návrh a realizaci algoritmu digitální stabilizace videa s cílem zajistit zpracování v reálném čase na platformě Raspberry Pi 4B. Digitální stabilizace je důležitá pro zlepšení vizuální kvality záznamu a spolehlivé zpracování obrazových dat v prostředích s omezenými výpočetními prostředky.

V práci jsou popsány základní principy digitální stabilizace a vybrané techniky počítačového vidění, které tvoří jádro implementace. Důraz byl kladen na jednoduchost a výpočetní efektivitu.

Implementace byla testována na zařízení Raspberry Pi 4B. Výsledky ukazují, že navržený přístup je schopný zajistit stabilizaci obrazu s nízkým zkreslením a dostatečnou plynulostí pro reálný čas, a to i za podmínek omezeného výkonu.

KLÍČOVÁ SLOVA

Algoritmy digitální stabilizace, Zpracování v reálném čase, Raspberry Pi 4B, počítačové vidění, vestavěné systémy.

ABSTRACT

This bachelor's thesis focuses on the design and implementation of a digital video stabilization algorithm intended for real-time processing on the Raspberry Pi 4B platform. Digital stabilization is essential for improving visual quality and enabling reliable image data processing in environments with limited computational resources.

The work outlines the basic principles of digital stabilization and selected computer vision techniques that form the core of the implementation. The focus was placed on simplicity and computational efficiency.

The implementation was tested on Raspberry Pi 4B. The results show that the proposed approach is capable of delivering stabilized video with low distortion and sufficient smoothness for real-time performance, even under constrained hardware conditions.

KEYWORDS

Digital Stabilization Algorithms, Real-Time Processing, Raspberry Pi 4B, computer vision, embedded systems.

NURKHAT, Alikhan. *Stabilizace videa na Raspberry Pi*. Bakalářská práce. Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav automatizace a měřicí techniky, 2025. Vedoucí práce: Ing. Dominik Řičánek

Prohlášení autora o původnosti díla

Jméno a příjmení autora:	Alikhan Nurkhat
VUT ID autora:	242251
Typ práce:	Bakalářská práce
Akademický rok:	2024/25
Téma závěrečné práce:	Stabilizace videa na Raspberry Pi

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 své bakalářské práce, panu Ing. Dominiku Řičánkovi, za odborné vedení, cenné konzultace, trpělivost a podnětné návrhy, které výrazně přispěly ke kvalitě této práce. Zvláště si vážím jeho vstřícného a lidského přístupu, který pro mě byl v průběhu zpracování práce velmi motivující a povzbudivý.
Velké poděkování patří také mé rodině, přítelkyni a spolužákům za jejich morální podporu během celého studia.

Obsah

Seznam obrázků

Seznam tabulek

Úvod

Digitální stabilizace videa je technika, jejímž cílem je potlačit nežádoucí chvění nebo vibrace obrazu způsobené neřízeným pohybem kamery během snímání. Tyto jevy mohou výrazně zhoršit nejen vizuální vnímání, ale i přesnost strojového zpracování videa. S rostoucím rozšířením přenosných zařízení, dronů a mobilní robotiky roste i potřeba stabilního obrazu, a to i v podmínkách, kde není možné využít složité mechanické stabilizátory.

Zatímco profesionální kamery často využívají optickou nebo mechanickou stabilizaci, v oblasti vestavěných systémů s omezeným výpočetním výkonem je nezbytné spoléhat na softwarová řešení. Digitální stabilizace zpracovává video po jednotlivých snímcích, analyzuje pohyb kamery a následně provádí korekci obrazu, čímž simuluje stabilní záznam bez nežádoucího pohybu.

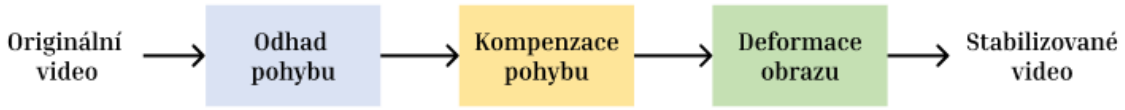
Tato bakalářská práce se zabývá návrhem, implementací a testováním algoritmu digitální stabilizace videa, který je schopen běžet v reálném čase na platformě Raspberry Pi 4B. Součástí práce je analýza metod počítačového vidění a rešerše existujících přístupů k digitální stabilizaci videa se zaměřením na výběr vhodných algoritmů pro zařízení s omezeným výpočetním výkonem. Navržena byla celková architektura systému, která byla následně implementována v jazyce Python. Výsledný algoritmus byl experimentálně vyhodnocen z hlediska kvality a výkonnosti přímo na cílovém zařízení.

1 Obecné principy metod digitální stabilizace videa

Digitální stabilizace videa (DVS – Digital Video Stabilisation) obvykle zahrnuje tři kroky. Nejprve se porovnáním dvou sousedních snímků videosekvence určí globální pohyb mezi snímky. Tento proces vyžaduje značné výpočetní zdroje, protože zahrnuje zpracování obrazu, což činí fázi odhadu pohybu nejsložitější a časově nejnáročnější. V tomto kroku se vytvoří model pohybu kamery a vypočítají se globální vektory pohybu.

Dalším krokem je korekce pohybu. V této fázi se od plánovaných pohybů kamery oddělí vysokofrekvenční nežádoucí oscilace. Na základě odfiltrovaných dat se vypočítá nová trajektorie pohybu. Pro kompenzaci chvění je snímkům přidělen pohyb v opačném směru než původní chvění, aby se získala stabilizovaná videosekvence.

V posledním kroku se provede deformace obrazu podle vypočítané stabilní trajektorie kamery [1, 9]. Tyto tři kroky lze považovat za samostatné fáze, které jsou postupně znázorněny na obrázku 1.1.



Obr. 1.1: Postup digitální stabilizace videa [1]

Offline metody stabilizace obvykle provádějí tyto kroky odděleně a zpracovávají každý z nich pro všechny snímky videa před přechodem k dalšímu kroku, zatímco online metody stabilizace je provádějí postupně pro každý snímek nebo malou sadu snímků [1].

Výsledkem digitální stabilizace je často mírně menší snímek než původní nestabilizovaný snímek. To je způsobeno tím, že během stabilizace dochází k posunům, rotacím a deformacím snímků, které kompenzují nežádoucí pohyby. Pro zachování stabilní centrální části se používá **ořez (cropping)**, což může vést k mírnému posunu centrální oblasti [10].

Při stabilizaci se navíc používá interpolace pixelů, která může způsobit rozmazání hran a ztrátu vysokofrekvenčních detailů. Tento efekt je nejvíce patrný na ostrých liniích a jemných texturách. Další nevýhodou digitální stabilizace je její vysoká

výpočetní náročnost. Moderní algoritmy však mohou být optimalizovány tak, aby umožňovaly stabilizaci v reálném čase [9].

1.1 Klasifikace algoritmů pro stabilizaci videa

Oblast digitální stabilizace videa prošla v posledních letech výraznými změnami díky zavedení metod hlubokého učení. Namísto explicitního výpočtu trajektorie kamery tyto metody využívají přístup učení s učitelem.

Podle zdroje [1] lze stávající metody digitální stabilizace videa zhruba rozdělit do dvou hlavních skupin podle použitého přístupu:

1. Klasické metody - založené na tradičních algoritmech pro zpracování obrazu a odhad pohybu kamery.
2. Metody hlubokého učení - využívají hluboké neuronové sítě, zejména konvoluční neuronové sítě (CNN), k automatické analýze a zpracování dat videa.

Podle toho, zda se zohledňují informace o hloubce, se digitální stabilizace videa dělí do tří skupin:

1. 2d stabilizace - předpokládá, že pohyb je omezen na dva rozměry.
2. 3D stabilizace - využívá 3D model scény pro přesnější analýzu složitých pohybů.
3. Hybridní stabilizace - kombinuje prvky 2D a 3D analýzy scény k dosažení kompromisu mezi přesností a výpočetní efektivitou.

1.2 Základní pojmy a principy stabilizace videa

V této části práce jsou popsány teoretické základy technik a algoritmů, které tvoří základ metod stabilizace videa rozebíraných v následujících kapitolách. Pochopení této části je zásadní pro porozumění klíčovým částem práce a navrhovaným řešením.

1.2.1 Konvoluce obrazu

Konvoluce obrazu [2] je základní operací při zpracování obrazu, která slouží k odstranění vysokofrekvenčního šumu, zvýšení ostrosti detailů, detekci hran či jiným modulacím frekvenční oblasti obrazu. Rovněž se využívá v neuronových sítích, například v konvolučních neuronových sítích (CNN). Konvoluce umožňuje upravovat hodnoty pixelů v obraze na základě jejich sousedů za použití speciální **konvoluční masky (konvolučního jádra)**.

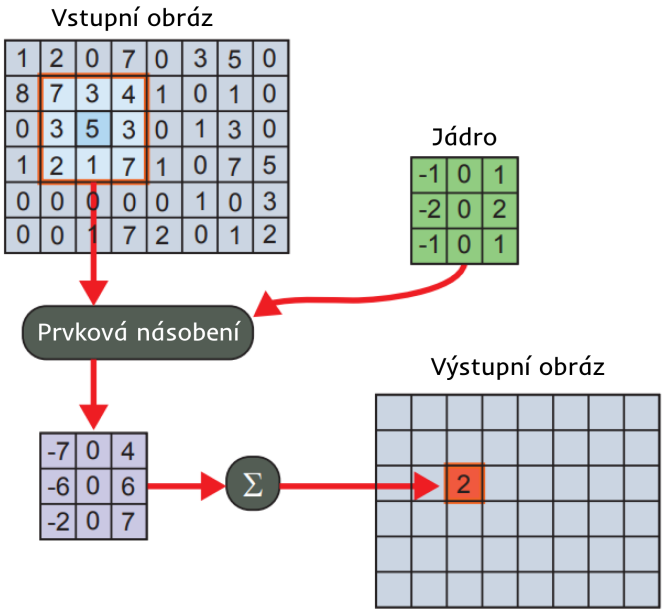
Pro 2D diskrétní obrazy je konvoluce definována jako skládání dvou funkcí (typicky matice): vstupního obrazu $y(i, j)$ a konvoluční masky $k(i, j)$:

$$(y * k)[i, j] = \sum_n \sum_m y[i - n, j - m]k[n, m] \tag{1.1}$$

V oblasti zpracování obrazu se korelace používá k porovnání šablony (masky) s určitým úsekem obrazu, aby se našla podobnost nebo specifické rysy. Korelace [11] se od konvoluce liší tím, že se u ní indexy masky „neotáčejí“. Její definice pro diskrétní 2D signál vypadá takto:

$$(y * k)[i, j] = \sum_n \sum_m y[i + n, j + m]k[n, m] \tag{1.2}$$

Pokud je jádro $k(x, y)$ symetrické, výsledek se od konvoluce neliší.



Obr. 1.2: Příklad s konvolučním jádrem 3×3 [2]

Obrázek 1.2 jasně ilustruje proces konvoluce pomocí malého jádra 3×3. Filtr je definován jako matice, kde prostřední prvek určuje váhu pro zpracovávaný pixel a ostatní prvky určují váhy pro sousední pixely. Příslušné pixely se vynásobí prvky jádra a jejich součet se zapíše do středového pixelu. Konvoluční jádro na obrázku 1.2 má poloměr 1, protože pokrývá nejbližší okolí centrálního pixelu (jeden kruh sousedů).

Při zpracování obrazových dat je nutné věnovat zvláštní pozornost pixelům na hranicích snímku. V okamžiku, kdy se konvoluční jádro dostává mimo oblast obrazu, je obvyklé považovat hodnoty neexistujících (vnějších) pixelů za nulové. Tento přístup sice umožňuje jednotné zpracování hraničních pixelů, avšak může vést ke vzniku

tmavého prstencového efektu podél okrajů. Tomuto nežádoucímu jevu lze předcházet oříznutím výsledného obrazu (ten se pak zmenší oproti původní velikosti) nebo replikací hraničních hodnot pixelů do vnější oblasti.

Konvoluce je užitečná, ale výpočetně náročná operace. Pro jádro o velikosti $k \times k$ je zapotřebí provést k^2wh násobení a sčítání, abychom zpracovali obraz o rozměrech $w \times h$. Některá konvoluční jádra však lze rozložit do dvou vektorů: jeden působí horizontálně a druhý vertikálně. Použití těchto dvou jader vede ke stejnému výsledku, avšak podstatně snižuje výpočetní složitost na $2kwh$ násobení a sčítání. Taková jádra se označují jako **separovatelná (separable kernels)**. Konvoluční jádro je považováno za separovatelné, pokud lze jeho matici K vyjádřit ve tvaru

$$K = u \cdot v^T \tag{1.3}$$

kde:

- u, v — vektory definující horizontální a vertikální složky.

Konvoluční jádro určuje efekt, který bude na obraz aplikován. V tomto článku budou uvedeny příklady jader pro výpočet gradientu, detekci rohů a hran, a vyhlazování šumu.

1.2.2 Metody vyhlazování

První metodou vyhlazování v této práci je **průměrovací filtr (Box Filter, Averaging)** [2]. Jedná se o nejjednodušší filtr s separovatelným jádrem, jehož výsledkem je aritmetický průměr centrálního a sousedních pixelů. Jádro průměrovacího filtru pro rozměry 3×3:

$$K = \begin{pmatrix} \frac{1}{9} & \frac{1}{9} & \frac{1}{9} \\ \frac{1}{9} & \frac{1}{9} & \frac{1}{9} \\ \frac{1}{9} & \frac{1}{9} & \frac{1}{9} \end{pmatrix}; \quad u = v = \begin{pmatrix} \frac{1}{3} & \frac{1}{3} & \frac{1}{3} \end{pmatrix} \tag{1.4}$$

Průměrovací filtr je jednoduchý, ale nevede k hladce rozmazanému obrazu (může způsobovat čtvercové artefakty). Místo něj se pro potlačení šumu častěji používá **Gaussovo rozmazání** [2], které je filtrem dolních frekvencí (potlačuje vysokofrekvenční složky). Dvourozměrná Gaussova funkce:

$$G(x, y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}} \tag{1.5}$$

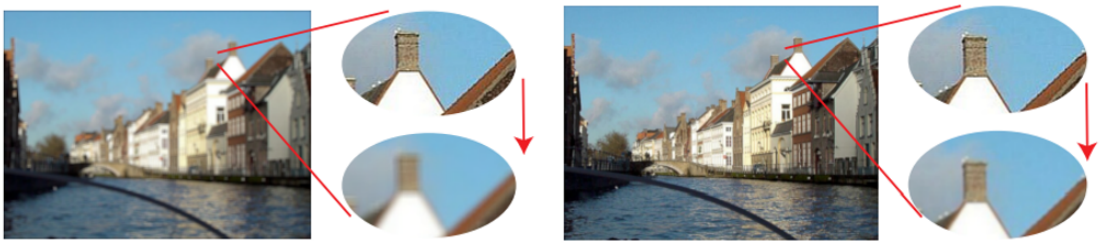
kde:

- σ — směrodatná odchylka Gaussovy funkce (čím větší σ, tím větší rozmazání),
- $G(x, y)$ — hodnota jádra Gaussova filtru v bodě (x, y).

Takto vypadá Gaussovo jádro o rozměrech 5×5:

$$K_{G_5} = \frac{1}{273} \begin{pmatrix} 1 & 4 & 7 & 4 & 1 \\ 4 & 16 & 26 & 16 & 4 \\ 7 & 26 & 41 & 26 & 7 \\ 4 & 16 & 26 & 16 & 4 \\ 1 & 4 & 7 & 4 & 1 \end{pmatrix}; \tag{1.6}$$

$$u = v = \begin{pmatrix} 0.061 & 0.242 & 0.383 & 0.242 & 0.061 \end{pmatrix}$$



Obr. 1.3: Porovnání výsledků průměrovacího filtru (vlevo) a Gaussova filtru (vpravo) při velikosti jádra 7×7 [2]

Obrázek 1.3 ukazuje srovnání průměrovacího filtru a Gaussova filtru pro vyhlazení obrazu. Jak je vidět, průměrovací filtr je jednoduchý, ale ve srovnání s Gaussovým filtrem vytváří hrubší rozostření.

1.2.3 Gaussova obrazová pyramida

Při sledování objektů ve videosekvenci je často nutné analyzovat klíčové body objektu v různých měřítkách. To umožňuje vybrat prvky, které jsou stabilní vůči změnám měřítka. Za tímto účelem se vytvoří sada snímků s různým rozlišením a uspořádá se do tvaru pyramidy: snímek s nejvyšším rozlišením se umístí do základny a snímek s nejnižším rozlišením na vrchol. Tato struktura se nazývá **obrazová pyramida (Image Pyramid)** [12].

Gaussova pyramida (Gaussian Pyramid)[12] je jedním z nejběžnějších typů obrazových pyramid. Původní obraz je na každé úrovni vyhlazen Gaussovým filtrem, aby se snížil šum a detaily. Poté se provede **podvzorkování (Downsampling)**: odstraní se každý druhý řádek a každý druhý sloupec obrazu. V důsledku toho se rozlišení v každé ose sníží na polovinu a plocha se zmenší čtyřnásobně. Tento přechod mezi úrovněmi pyramidy se nazývá **oktáva (Octave)**.

Proces vyhlazování a snižování rozlišení se opakuje na každé další úrovni a s každým dalším krokem se rozlišení obrazu stále snižuje a detaily jsou méně výrazné.

Generování úrovní pyramidy pokračuje, dokud se obraz nezmenší na velikost, při které další zpracování ztrácí smysl. Například když velikost Gaussova konvolučního jádra přesáhne velikost obrazu na jedné ze stran.

1.2.4 Detektor rohů FAST

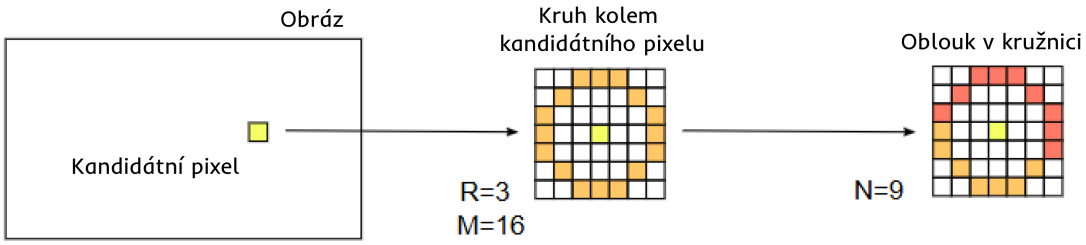
Obr. 1.4: Výsledky detekce rohů pomocí FAST [3]

Features from Accelerated Segment Test (FAST) [3] je výpočetně efektivní algoritmus pro detekci rohů ve vstupním obraze, jehož výstupem jsou souřadnice klíčových bodů. Pro každý zkoumaný pixel p je vytvořena kružnice o poloměru $R = 3$ pixely obsahující $M = 16$ pixelů. Intenzita středového pixelu je označena jako I_p. Pokud je alespoň N po sobě jdoucích pixelů (oblouků) na kružnici světlejších nebo tmavších než centrální pixel I_p o prahovou hodnotu t, je zkoumaný pixel považován za rohový. Podobným způsobem se testují všechny pixely vstupního obrazu.

Pro rychlé vyhledávání nevhodných kandidátů [13] se nejprve kontrolují čtyři pixely (nahoře, dole, vlevo, vpravo). Pokud jsou alespoň tři z nich světlejší nebo tmavší než středový pixel o prahovou hodnotu t, zahájí se úplná kontrola. Pokud předběžná kontrola selže, je pixel vyřazen a algoritmus přejde k dalšímu. Mechanismus odmítnutí však nemusí fungovat správně, pokud $N < 12$.

Obrázek 1.6 ukazuje příklad zkoumaného pixelu (zvýrazněného žlutě) a oblouku (zvýrazněného červeně) na kružnici kolem něj (zvýrazněné oranžově). V tomto příkladu je délka oblouku $N = 9$ (standardní konfigurace FAST-9).

Pro snížení počtu nadbytečných rohů se používá test **potlačení nemaxim (non-maximum suppression)**, který snižuje počet klíčových bodů a ponechává pouze ty nejvýraznější, avšak zároveň snižuje výkon FAST kvůli dalším výpočtům. Každý pixel označený jako roh zkontroluje své nejbližší okolí (prstenec o poloměru 1 pixelu) na přítomnost dalších rohů. Pokud se v sousedství nachází dva nebo více rohů, pro každý z nich se vypočítá hodnota V, která se rovná součtu absolutních



Obr. 1.5: Ukázka algoritmu FAST [3]

rozdílů mezi hodnotou p a 16 pixely na tomto prstenci. Rohy s nižší než maximální hodnotou V jsou následně vyřazeny.

Prahová hodnota t určuje citlivost algoritmu. Nízká prahová hodnota zvyšuje počet detekovaných rohů, ale činí algoritmus citlivým na šum a může vést k falešně pozitivním výsledkům. Vysoká prahová hodnota snižuje počet rohů, čímž se algoritmus stává robustnějším, ale může přehlédnout slabé rohy.

Algoritmus FAST je mnohem rychlejší než jiné detektory rohů, ale jeho citlivost a odolnost vůči šumu silně závisí na parametru t.

1.2.5 Detektor rohů Harrise

Obr. 1.6: Výsledky detekce úhlů Harrisovou metodou [4]

Harrisův rohový detektor [14] je jednou z prvních metod detekce rohových bodů v obraze. Rohové body v obraze jsou oblasti, kde se intenzita pixelů mění ve všech směrech. Metoda vyhodnocuje změnu intenzity obrazu při malých posunech v různých směrech a oblasti s maximálními změnami identifikuje jako rohy.

Nejprve se obraz v předzpracování vyhladí Gaussovým filtrem, aby se snížil šum. Poté se použije **Sobelův filtr**, což jsou konvoluční masky pro výpočet horizontálních a vertikálních gradientů [4]. Pro konvoluční masku o velikosti 3×3 vypadá Sobelův filtr takto

$$sobel_x = \frac{1}{4}\begin{pmatrix}1\\2\\1\end{pmatrix} \cdot \begin{pmatrix}-1 & 0 & 1\end{pmatrix}; \qquad sobel_y = (sobel_x)^T \tag{1.7}$$

Sestaví se strukturní tenzorová matice, která obsahuje informace o rozložení gradientů v okolí pixelu.

$$M = \sum_{p \in B} \begin{pmatrix} I_x^2(p) & I_x(p)I_y(p) \\ I_x(p)I_y(p) & I_y^2(p) \end{pmatrix} \tag{1.8}$$

kde:

- p — souřadnice pixelů v okně B o velikosti 3×3,
- $I(p)$ — funkce vstupního obrázku,
- $I_x(p)$ — horizontální gradient, kde $I_x(p) = I(p) * sobel_x$,
- $I_y(p)$ — vertikální gradient, kde $I_y(p) = I(p) * sobel_y$.

Nyní je třeba spočítat Harrisovu odezvu s použitím koeficientu citlivosti:

$$R = det\,(M) - k\,(trace\,(M))^2 \tag{1.9}$$

kde:

- $\det(M) = \lambda_1\lambda_2$ — determinant matice M,
- $\text{trace}(M) = \lambda_1 + \lambda_2$ — stopa matice M,
- λ_1, λ_2 — vlastní čísla matice M,
- k — koeficient citlivosti.

Pokud je hodnota R velká a kladná, je bod považován za rohový. Čím větší je hodnota R, tím je **roh výraznější (Corner Strength)**.

Existuje malé vylepšení klasického Harrisova úhlového detektoru s názvem Shi-Tomasi, které nabízí:

$$R \;=\; min(\lambda_1,\ \lambda_2) \tag{1.10}$$

Pokud je R větší než určitá prahová hodnota, znamená to, že byl detekován roh.

1.2.6 Deskriptor příznaků BRIEF

Deskriptor příznaků [15] – metoda popisu lokálních příznaků, používaná v počítačovém vidění pro vyhledávání a rozpoznávání objektů na obrázcích. Popisuje lokální

charakteristiky (gradienty, textury, kontury) kolem zadaného bodu, což umožňuje efektivní porovnávání a přiřazování příznaků mezi různými obrázky.

BRIEF (Binary Robust Independent Elementary Features) [16] – je binární deskriptor příznaků. Na předem vyhlazeném obrazu se kolem zajímavého bodu vytvoří oblast p o velikosti $S \times S$. Uvnitř této oblasti se vybere pevná sada párů pixelů $n_d(x, y)$ (n_d je standardně 256, ale může být také 128 nebo 512); pixely se vybírají náhodně uvnitř oblasti. Pro každý pár se provádí test intenzity τ vyjádřený jako

$$\tau(p; x, y) := \begin{cases} 1, & p(x) < p(y) \\ 0, & p(x) \geq p(y) \end{cases} \tag{1.11}$$

kde:

- $p(x)$ a $p(y)$ — intenzity pixelů v páru.

Výsledky všech testů se spojí do n_d-bitového řetězce, kde každý bit odpovídá jednomu testu intenzity. Výsledný řetězec se vytvoří podle vzorce:

$$f_{n_d}(p) := \sum_{i=1}^{n_d} 2^{i-1} \tau(p; x_i, y_i) \tag{1.12}$$

Porovnávání deskriptorů se provádí počítáním rozdílných pozic ve dvou bitových řetězcích x, y stejné délky n

$$H_{xy} = \sum_{i=1}^{n} x_i \oplus y_i \tag{1.13}$$

kde H_{xy} se nazývá **Hammingova vzdálenost (Hamming Distance)** [17]. Pokud je vzdálenost menší než určitá prahová hodnota, deskriptory se považují za shodné.

Je důležité si uvědomit, že BRIEF není detektor klíčových bodů, ale pouze popisuje lokální vlastnosti předem vybraného klíčového bodu (např. pomocí FAST).

1.2.7 Algoritmus ORB

Klíčové body, které představují lokální vlastnosti obrazu (rohy, hrany), by měly být odolné vůči změnám osvětlení a invariantní vůči otáčení a změnám měřítka. Mezi nejpopulárnější algoritmy pro detekci klíčových bodů patří: SIFT (Scale-Invariant Feature Transform), SURF (Speeded-Up Robust Features) a **ORB (Oriented FAST and Rotated BRIEF)** [18].

V tomto seznamu má ORB výhodu, protože je relativně jednoduchý a nenáročný na výpočetní prostředky, což jej činí vhodným pro použití v reálném čase. Hlavní nevýhodou ORB je však nižší spolehlivost při popisu vlastností obrazů, které se mění

z hlediska úhlu a měřítka. Ačkoli je ORB navržen pro zpracování takových případů, v tomto ohledu není tak efektivní jako jiné algoritmy, například SIFT nebo SURF.

Algoritmus ORB se skládá z následujících kroků:

1. Vytvoření Gaussovy pyramidy: Z obrazu je vytvořena Gaussova pyramida, kde na každé úrovni pyramidy je použit algoritmus FAST pro detekci rohů.
2. Detekce rohů Harrise: Pro každý nalezený roh je aplikován Harrisův detektor rohů, který vypočítává skóre rohů (cornerness score) pomocí okna 3×3 a koeficientu citlivosti rovného 1.
3. Seřazení a výběr nejlepších bodů: Všechny nalezené rohy jsou seřazeny podle hodnoty skóre rohů a je vybráno pouze N nejlepších bodů s nejvyšším skóre. Zůstávají tak pouze nejvýraznější rohy. Algoritmus ORB může vynechat kroky 2 a 3 a používat pouze detektor rohů FAST, což zrychluje jeho činnost, avšak snižuje kvalitu detekovaných klíčových bodů.
4. Seskupení rohů z jednotlivých úrovní pyramidy: Rohy z každé úrovně pyramidy jsou shromážděny do jednoho pole, kde každý roh obsahuje hodnoty $(x, y, octave)$ kde: *octave* - úroveň pyramidy, na které byl roh nalezen, (x, y) - pozice uvnitř obrazu na této úrovni pyramidy. Po dosažení maximální kapacity pole mohou být rohy z nejvyšších úrovní pyramidy vyřazeny.
5. Určení úhlu orientace: ORB určuje úhel orientace každého klíčového bodu, aby byl deskriptor invariantní vůči rotaci. Úhel orientace je počítán pouze pro klíčové body ze základní úrovně s nejvyšším rozlišením pomocí centroidu intenzity (Intensity Centroid) na základě čtvercové oblasti kolem detekovaného rohu. Úhel orientace se vypočítá následovně:

$$\theta = arctan2\left(m_{01}, m_{10}\right) \tag{1.14}$$

   kde:

   - $m_{10} = \sum x \cdot I(x, y)$ a $m_{01} = \sum y \cdot I(x, y)$ — momenty intenzity obrazu, přičemž součet se počítá pro všechny pixely v dané oblasti,
   - $I(x, y)$ — intenzita obrazu v bodě (x, y).

6. Po vypočítání orientace klíčových bodů na základní úrovni je pro každý klíčový bod vytvořen deskriptor BRIEF s invariantností vůči rotaci (rBRIEF). Před vytvořením řetězce o délce 256 bitů je pro každý klíčový bod ze všech úrovní pyramidy šablona pro výběr párů pixelů otočena o úhel θ kolem osy Z, aby odpovídala orientaci klíčového bodu. To se provádí následujícím způsobem:

$$(x', y') = (x \cdot cos(\theta) - y \cdot sin(\theta), x \cdot sin(\theta) + y \cdot cos(\theta)) \tag{1.15}$$

   kde:

   - (x, y) — původní souřadnice bodu ze šablony,

- (x', y') — nové souřadnice bodu po otočení o úhel θ.

1.2.8 Porovnávání klíčových bodů

Po extrakci a popisu klíčových bodů mezi některými sekvencemi snímků je nutné porovnat (spárovat) deskriptory mezi sousedními snímky, aby bylo možné zrekonstruovat trajektorii pohybu kamery.

Klíčové body lze porovnávat jednoduchým procházením všech možností. Tato metoda porovnává deskriptor z jednoho obrázku se všemi deskriptory z druhého obrázku a hledá pár s nejmenší vzdáleností. Porovnávání se provádí pomocí Hammingovy vzdálenosti. Tento přímý způsob porovnávání klíčových bodů se nazývá **Brute Force Matcher (BFM)** [19].

Pro zvýšení přesnosti porovnávání lze přidat **křížovou kontrolu (Cross-Check)**. Každý klíčový bod z prvního snímku se páruje s nejlepším odpovídajícím bodem z druhého snímku. Následně se provede zpětné porovnání: bod z druhého snímku hledá nejlepší odpovídající bod v prvním snímku. Pokud si oba body vzájemně odpovídají, je spárování považováno za správné a uchováno.

Tato metoda je jednoduchá a spolehlivá při malém množství klíčových bodů, ale může být nepřijatelně pomalá pro snímky s vysokým rozlišením a velkým množstvím klíčových bodů.

1.2.9 Algoritmus RANSAC

Při párování klíčových bodů mohou vlivem podobných textur, šumu a změn osvětlení vznikat falešné shody, kdy jsou různé objekty či oblasti obrazu chybně označeny jako odpovídající body. Tyto falešné shody vedou k výskytu **odlehlých bodů (outliers)** – bodů, které neodpovídají společnému vztahu mezi dvěma množinami dat.

RANdom SAmple Consensus (RANSAC) [20] je algoritmus určený k odhadu parametrů modelu v přítomnosti odlehlých bodů. V počítačovém vidění je tento přístup velmi oblíbený, zejména při řešení úloh, jako je stabilizace videa.

RANSAC vytváří kandidátní řešení na základě minimálního počtu pozorování (bodů), který je potřebný k odhadu parametrů modelu.

Algoritmus RANSAC provádí následující kroky:

1. Náhodně se vybere minimální počet bodů m, potřebných pro sestavení modelu.
2. Na základě vybraných bodů se sestaví hypotetický model.
3. Stanoví se, kolik bodů z celé datové sady odpovídá sestavenému modelu s určitou dovolenou chybou ϵ. Bodům, které jsou v souladu s modelem, se říká **inliers (souladné body)**.

4. Pokud poměr počtu soulavných bodů (inliers) k celkovému počtu bodů v datové sadě překročí zadaný práh τ, parametry modelu se přepočítají s využitím všech nalezených inliers, čímž se získá přesnější model a algoritmus skončí.
5. V opačném případě se kroky 1–4 opakují N krát. Model s nejvyšším počtem inliers je zvolen jako nejlepší model.

Maximální počet iterací N se vybírá tak, aby s danou pravděpodobností p bylo zaručeno, že alespoň jedna z iterací zvolí podmnožinu bodů sestávající výhradně z inliers. Počet iterací lze spočítat podle vzorce:

$$N = \frac{log\,(1-p)}{log\,(1-(1-v)^m)} \tag{1.16}$$

kde:

- p — pravděpodobnost, že alespoň v jedné iteraci bude vybrána podmnožina bodů obsahující pouze inliers (obvykle se volí 0.99),
- v — pravděpodobnost, že náhodně vybraný bod je outlier,
- m — minimální počet bodů nutný pro sestavení modelu.

V této práci se RANSAC používá pro afinní model, který vyžaduje minimálně tři náhodné body. Metrikou odchylky od daného modelu je euklidovská vzdálenost:

$$d = \sqrt{(x-x')^2 + (y-y')^2} \tag{1.17}$$

kde:

- (x, y) — skutečná pozice,
- (x', y') — pozice bodu predikovaná afinním modelem.

1.2.10 Rozklad na bitové roviny

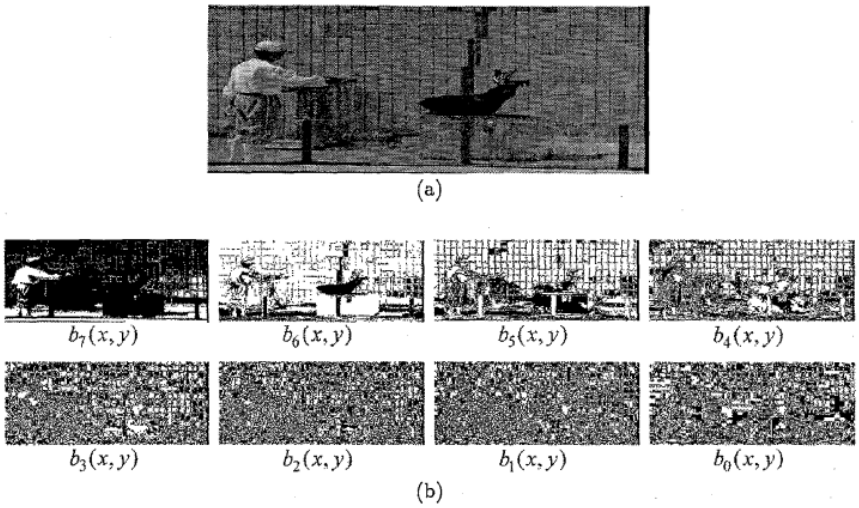
Rozklad bitové roviny [5] je způsob, jak rozložit jas pixelů na jednotlivé úrovně významnosti. Vyšší bity jsou zodpovědné za hrubé detaily a celkovou strukturu, zatímco nižší bity jsou zodpovědné za jemné detaily a šum. Tento přístup zjednodušuje zpracování obrazu a umožňuje přesnější a efektivnější filtrování a analýzu.

Intenzita každého pixelu v obrazech ve stupních šedi je zakódována v binární podobě (např. 8 bitů), kde každý bit odráží určitou úroveň jasu:

$$f^t\,(x,\ y) = a_{K-1} \cdot 2^{K-1} + a_{K-2} \cdot 2^{K-2} + \ldots + a_1 \cdot 2^1 + a_0 \cdot 2^0 \tag{1.18}$$

kde:

- $a_k,\ 0 \leq k \leq K$ — k-tý bit, který může nabývat hodnot 0 nebo 1,
- K — hloubka diskretizace,
- $f^t(x, y)$ — intenzita pixelu v bodě (x, y).



Obr. 1.7: Příklad rozkladu obrazu ve stupních šedi. (a) Původní obraz (b) Výsledek rozkladu na bitové roviny [5]

Bitové roviny jsou sady těchto bitů o stejné velikosti pro všechny pixely. Vyšší bitové roviny obsahují informace o velkých strukturách obrazu a jeho celkovém jasu, zatímco nižší představují jemnější detaily včetně šumu (obrázek 1.7). Tento rozklad zjednodušuje analýzu obrazu, protože je možné uvažovat jeho úrovně významnosti odděleně.

1.2.11 Afinní transformace

Afinní transformace [21, 22] jsou typem lineárních transformací, které zachovávají přímočarost (přímky zůstávají přímkami), rovnoběžnost (rovnoběžné přímky zůstávají rovnoběžnými) a proporce (poměr bodů na jedné přímce), ale nemusí nutně zachovávat úhly a délky. Metoda se obvykle používá pro korekci geometrických zkreslení a deformací vznikajících při nedokonalých úhlech snímání, což činí tuto metodu široce použitelnou při zpracování obrazu a také jedním z klíčových prvků při stabilizaci videa.

Afinní transformaci lze obecně popsat takto:

$$q = Ap + b \tag{1.19}$$

kde:

- $p \in \mathbb{R}^n$ — vstupní vektor (bod v n-rozměrném prostoru),
- A — lineární transformační matice rozměru $n \times n$, která odpovídá za rotaci, škálování a sklon,
- $b \in \mathbb{R}^n$ — vektor posunutí odpovídající za translaci,
- $q \in \mathbb{R}^n$ — výstupní vektor reprezentující bod v nové poloze.

V případě stabilizace videa před sestavením afinního modelu pohybu předem známe vstupní a výstupní vektor, které reprezentují sadu klíčových bodů ve dvou po sobě jdoucích snímcích. Afinní transformace pro snímek $t-1$ a t pak vypadá takto:

$$\begin{pmatrix} x_t \\ y_t \end{pmatrix} = \begin{pmatrix} a_{00} & a_{01} \\ a_{10} & a_{11} \end{pmatrix} \begin{pmatrix} x_{t-1} \\ y_{t-1} \end{pmatrix} + \begin{pmatrix} b_{00} \\ b_{10} \end{pmatrix} \tag{1.20}$$

kde:

- $(x_t, y_t)\, a\, (x_{t-1}, y_{t-1})$ - souřadnice klíčového bodu ve snímcích t a $t-1$,
- b_{00}, b_{10} - koefcienty translace v horizontálním a vertkálním směru,
- a_{00}, a_{11} - koeficienty škálování (měřítka) v horizontálním a vertikálním směru,
- a_{01}, a_{10} - vyjadřují posun a rotaci.

Homogenizace pak umožňuje zapsat afinní transformaci sloučením do jediné matice M:

$$q \;=\; Mp \iff \begin{pmatrix} x_t \\ y_t \\ 1 \end{pmatrix} = \begin{pmatrix} a_{00} & a_{01} & b_{00} \\ a_{10} & a_{11} & b_{10} \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x_{t-1} \\ y_{t-1} \\ 1 \end{pmatrix} \tag{1.21}$$

Neznámá matice M vytváří afinní transformaci, která popisuje, jak se klíčový bod v snímku $t-1$ transformuje na klíčový bod ve snímku t. Rovnice 1.21 obsahuje 6 neznámých proměnných, a proto je k jejímu řešení potřeba vzít minimálně 3 odpovídající si dvojice klíčových bodů ve dvou po sobě jdoucích snímcích.

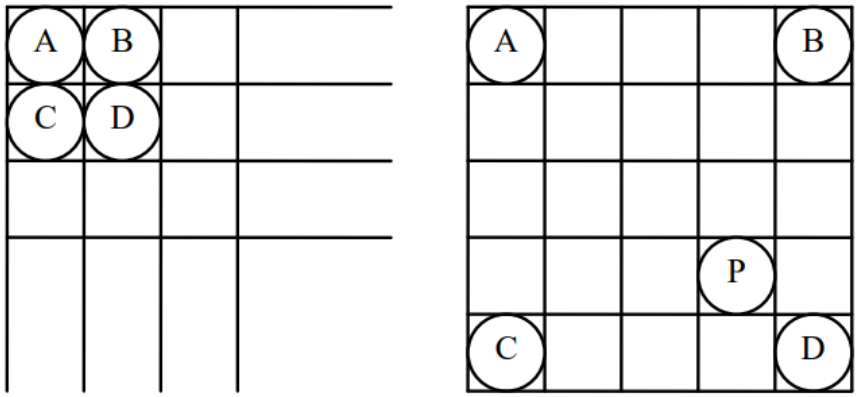
Přepsáním rovnice 1.21 pro $\left(x_{t-1}^{(i)}, y_{t-1}^{(i)}\right) \rightarrow \left(x_t^{(i)}, y_t^{(i)}\right), i = 1, 2, \ldots, N$ kde $N \geq 3$ v maticovém tvaru:

$$\underbrace{\begin{pmatrix} x_{t-1}^{(1)} & y_{t-1}^{(1)} & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & x_{t-1}^{(1)} & y_{t-1}^{(1)} & 1 \\ x_{t-1}^{(2)} & y_{t-1}^{(2)} & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & x_{t-1}^{(2)} & y_{t-1}^{(2)} & 1 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ x_{t-1}^{(N)} & y_{t-1}^{(N)} & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & x_{t-1}^{(N)} & y_{t-1}^{(N)} & 1 \end{pmatrix}}_{2N \times 6} \underbrace{\begin{pmatrix} a_{00} \\ a_{01} \\ b_{00} \\ a_{10} \\ a_{11} \\ b_{10} \end{pmatrix}}_{6 \times 1} = \underbrace{\begin{pmatrix} x_t^{(1)} \\ y_t^{(1)} \\ x_t^{(2)} \\ y_t^{(2)} \\ \vdots \\ x_t^{(N)} \\ y_t^{(N)} \end{pmatrix}}_{2N \times 1} \tag{1.22}$$

Matici vstupních dat o rozměrech $2N \times 6$ označme jako x, výstupní vektor o rozměrech $2N \times 1$ označme jako y. Následně použijeme metodu nejmenších čtverců pro výpočet parametrů vektoru M:

$$M = \left(x^T x\right)^{-1} x^T y \tag{1.23}$$

Tímto způsobem afinní transformace efektivně popisují pohyby kamery mezi snímky a poskytují přesné modelování dvourozměrného pohybu při nízké výpočetní náročnosti.



Obr. 1.8: Schematický diagram zvětšení obrazu [6]

1.2.12 Interpolace

Interpolace při zpracování obrazu je jedním z klíčových nástrojů, který se často používá pro zvětšení měřítka obrazu [6]. Jednoduše řečeno, když je potřeba převést obrázek s nízkým rozlišením na ostřejší a detailnější verzi, na pomoc přicházejí interpolační algoritmy. Každý z těchto algoritmů má své silné a slabé stránky a právě na výběru metody závisí, jak kvalitní bude výsledný obraz.

Na obrázku 1.8 je vidět, jak jsou pixely rozloženy při zvětšení obrazu čtyřikrát. Původní pixely, označené jako A, B, C a D, obklopují prázdné oblasti, které jsou vyplněny novými pixely (například pixel P). Hodnota jasu nebo barvy těchto nových bodů se vypočítává pomocí určité interpolační metody. To, která metoda bude použita, závisí na zadaném úkolu a požadované úrovni detailů.

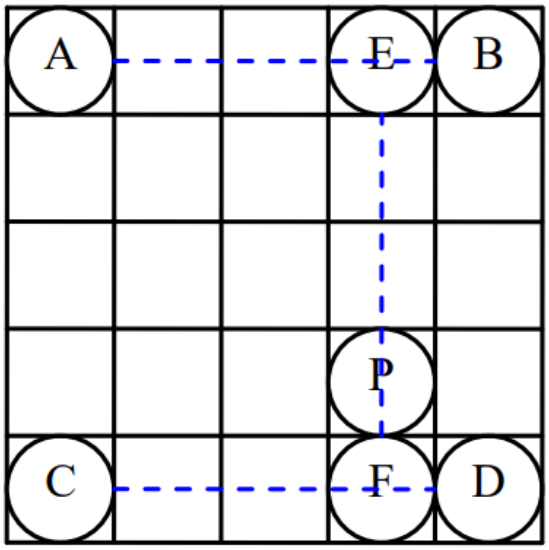
A. Interpolace metodou nejbližšího souseda// Algoritmus určuje nejbližší pixel v původním obraze pro každý pixel zvětšeného obrazu a přiřazuje mu jeho hodnotu bez dalších výpočtů. Tato metoda je nejjednodušší a nejrychlejší, avšak může vést k výrazným zkreslením a artefaktům.

B. Bilineární interpolace

Poloha pixelu P ve zvětšeném obraze je převedena do původního obrazu, poté se vypočítá vliv čtyř pixelů A, B, C, D. Čím blíže je pixel, tím větší má vliv na hodnotu pixelu P.

Předpokládá se, že souřadnice pixelů A, B, C, D odpovídají bodům $(i, j), (i, j + 1), (i + 1, j), (i + 1, j + 1)$. Souřadnice pixelu P je přiřazena bodu (u, v). Následně je bilineární interpolace provedena podle následujících kroků:

1. Vliv pixelů A a B je vypočítán, přičemž na obrázku 1.9 je označen jako E, a



Obr. 1.9: Diagram bilineárního interpolačního algoritmu [6]

je určen následujícím způsobem:

$$f\left(i, j+v\right) = \left[f\left(i, j+1\right) - f\left(i, j\right)\right] v + f\left(i, j\right) \tag{1.24}$$

2. Vliv pixelů C a D je vypočítán, přičemž je označen jako F, a je určen následujícím způsobem:

$$f\left(i+1, j+v\right) = \left[f\left(i+1, j+1\right) - f\left(i+1, j\right)\right] v + f\left(i+1, j\right) \tag{1.25}$$

3. Hodnota pixelu P je vypočítána na základě vlivů E a F:

$$\begin{aligned} f(i+u, j+v) = (1-u)(1-v) f(i,j) - (1-u) v f(i, j+1) \\ + u(1-v) f(i+1, j) + uv f(i+1, j+1) \end{aligned} \tag{1.26}$$

Bilineární interpolace je ve srovnání s metodou nejbližšího souseda složitější a vyžaduje více výpočetních prostředků. Odstraňuje však diskrétní přechody mezi pixely a zajišťuje plynulejší gradienty. Díky svým vlastnostem nízkofrekvenční filtrace může bilineární interpolace vyhlazovat obrysy obrazu, což někdy vede k rozmazání hran. Tato metoda představuje rozumný kompromis mezi výkonem a kvalitou.

C. Bikubická interpolace

Bikubická interpolace je rozšířením bilineární interpolace. Zatímco bilineární interpolace zohledňuje 4 sousední pixely, bikubická interpolace pracuje s 16 sousedními pixely, což umožňuje dosáhnout plynulejších a přirozenějších přechodů mezi pixely.

Při zvětšení obrazu je pro každý nový pixel P vybrána oblast složená z 16 nejbližších pixelů (mřížka 4×4). Jejich hodnoty intenzity jsou použity pro interpolaci. Hodnota nového pixelu je vypočítána pomocí dvou po sobě jdoucích jednorozměrných kubických interpolací:

1. Nejprve se provede interpolace pro každý řádek ze čtyř pixelů.
2. Poté se získané mezihodnoty interpolují po sloupcích, aby se získala konečná hodnota pixelu.

Funkce kubické interpolace může být vyjádřena následujícím vzorcem:

$$f(x) = a_0 + a_1 x + a_2 x^2 + a_3 x^3 \tag{1.27}$$

Koeficienty a_0, a_1, a_2, a_3 jsou určeny na základě hodnot sousedních pixelů a jejich derivací.

Bikubická interpolace poskytuje nejostřejší kvalitu obrazu, ale vyžaduje výrazně více výpočetních prostředků.

2 Klasické metody digitální stabilizace

V počátcích digitální stabilizace videa se nejčastěji používaly dvourozměrné transformace – především afinní a projektivní. Byly vhodné pro horizontální a vertikální posuny, rotace a škálování, relativně snadno se implementovaly a zároveň byly dostatečně rychlé pro práci v reálném čase. Tento přístup má však závažnou nevýhodu: nezohledňuje paralaxu, která vzniká při pohybu kamery dopředu nebo dozadu vzhledem k natáčené scéně. To znamená, že objekty v různých vzdálenostech od kamery se posouvají rozdílně a pod různými úhly, takže dvourozměrný model pohybu takové zkreslení nemůže korigovat [1].

Pro řešení tohoto problému se objevily metody trojrozměrné stabilizace, které rekonstruují nejen projekci pohybu, ale také celou geometrii scény. Umožňují kompenzovat paralaxu a získat rovnoměrnější a realističtější obraz. Plnohodnotné 3D modelování je však složitý proces náročný na zdroje, který vyžaduje velké výpočetní náklady a čas. Navíc při špatném osvětlení, nebo pokud některé objekty v záběru zakrývají jiné, může být 3D model sestaven nesprávně [23].

Hybridní metody byly vyvinuty jako kompromis mezi rychlostí a kvalitou výsledku. „Zdokonalují“ klasické 2D modely umělou korekcí trajektorie pohybu tak, aby se přiblížily výsledkům 3D metod. Tento přístup však často přináší znatelná zkreslení a výsledná stabilizace ne vždy odráží skutečnou geometrii scény [23].

Navzdory těmto omezením je 2D pohybový model využívající afinní transformace stále nejoblíbenějším řešením v mnoha aplikacích. Poskytuje vysokou výpočetní rychlost, snadnou implementaci a dostatečnou přesnost pro scénáře, kde paralaxa nehraje rozhodující roli [24].

V této kapitole jsou proto popsány dvourozměrné modely pohybu, zejména afinní transformace, které s přijatelnou přesností umožňují popisovat posuny, rotace a škálování, a současně se vyznačují nízkou výpočetní náročností.

2.1 Fáze odhadu pohybu

Analýza pohybu je prvním a klíčovým krokem v procesu digitální stabilizace videa. Robustnost a přesnost výsledného stabilizovaného videa závisí na kvalitě odhadu pohybu. Navíc tato fáze vyžaduje značné výpočetní zdroje a čas, protože je spojena se zpracováním velkého množství dat získaných z video streamu. Proto se výběr optimálního algoritmu pro odhad pohybu stává klíčovým úkolem, kde je důležité najít rovnováhu mezi rychlostí, přesností a výpočetní efektivitou [25].

V reálném čase se analyzuje vstupní videosekvence. Při určité frekvenci vzorkování jsou do systému přiváděny po sobě jdoucí snímky $t-1$ a t (kde $t = 2,\ 3,\ 4, \ldots,\ N, N$ - maximální počet snímků). Pomocí metod odhadu pohybu se

vytváří sada vektorů V_t které popisují, jak se objekty nebo oblasti obrazu posunuly mezi snímky $t-1$ a t [9].

Nicméně původní sada vektorů často obsahuje chyby způsobené šumem, změnami osvětlení nebo artefakty zpracování obrazu. Takové vektory výrazně odbočují od celkové struktury dat. K jejich filtrování se používá algoritmus RANSAC, který pomáhá vyloučit podobné odlehlé body.

Filtrovaná sada vektorů V_t nyní obsahuje vektory, které odpovídají předběžnému afinnímu modelu pohybu vytvořenému algoritmem RANSAC.

S využitím metody nejmenších čtverců se ze sady V_t vypočítá přesnější afinní matice T_t, která popisuje, jak se snímek $t-1$ transformoval na snímek t.

$$T_t = \begin{pmatrix} a_{00} & a_{01} & b_{00} \\ a_{10} & a_{11} & b_{10} \\ 0 & 0 & 1 \end{pmatrix} \tag{2.1}$$

kde:

- b_{00}, b_{10} - posun podle os X a Y,
- $\theta = arctan2\,(a_{01}a_{00})$ úhel rotace snímku.

Tato matice se stává základem pro následující fázi — vyhlazení trajektorie pohybu, což umožňuje minimalizovat vizuální artefakty a dosáhnout plynulosti stabilizace.

Existuje široká škála algoritmů pro odhad pohybu v kontextu stabilizace videa, a proto není smysluplné zkoumat všechny z nich. Místo toho se tato práce zaměřuje na metody, které vykazují uspokojivé výsledky v reálném čase a při omezených výpočetních zdrojích.

V této kapitole jsou podrobně probrány následující metody:

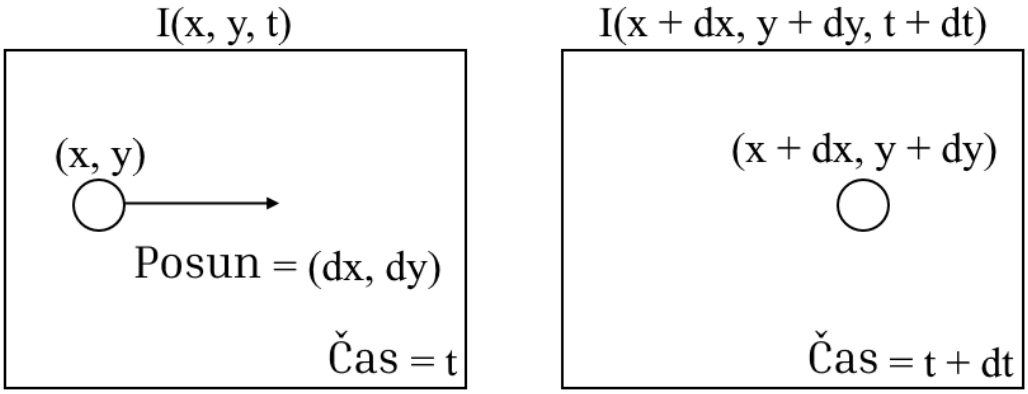
1. Algoritmus Lucase-Kanade pro sledování optického toku,
2. Blokové porovnávání s dekompozicí na bitové vrstvy,
3. Porovnávání klíčových bodů s detektorem ORB.

Metody založené na porovnávání klíčových bodů a optickém toku jsou klasickými přístupy pro odhad pohybu ve videu. Vyznačují se relativně nízkými výpočetními nároky, protože analyzují pohyb na úrovni malých částí obrazu a vyhýbají se globálním výpočtům. Metoda blokového porovnávání byla zvolena kvůli relativně jednoduché implementaci, avšak v čisté podobě může být algoritmus výpočetně náročný. Pokud se však aplikuje dekompozice na bitové vrstvy, stává se tento algoritmus efektivním pro vestavěné systémy.

V této práci nebudou zvažovány metody, které vyžadují výpočet pohybu pro každý pixel jednotlivě nebo aplikaci transformací na celý obraz, protože výpočetní složitost těchto metod roste exponenciálně s velikostí vstupního obrazu. Dále tato

práce nezahrnuje metody využívající více kamer nebo kamery s hloubkovým snímáním, které jsou potřebné pro obnovení plné trajektorie pohybu kamery v 3D a které mohou zohledňovat složité pohyby a perspektivní zkreslení, avšak nejsou použitelné v kontextu této práce. Rovněž nejsou zvažovány řady metod, které se spoléhají na data z gyroskopů a akcelerometrů pro obnovení pohybu kamery.

2.1.1 Algoritmus Lucase-Kanade pro sledování optického toku



Obr. 2.1: Problém optického toku [7]

Metoda optického toku je klasickým přístupem k úlohám sledování pohybu objektů a kamery na základě sekvenčních snímků.

Optický tok je pohyb objektů mezi sekvenčními snímky, způsobený relativním pohybem objektu a kamery [26]. Tato metoda je založena na dvou předpokladech: 1) Intenzita pixelu objektu na snímku zůstává nezměněná při jeho pohybu mezi sekvenčními snímky 2) Sousední pixely se pohybují po jedné trajektorii. Tato úloha je dobře ilustrována na obrázku 2.2.

Obraz lze představit jako funkci intenzity pixelů, která závisí na prostorových souřadnicích a čase. Označme levou část obrázku 2.2 jako $I(x, y, t)$. Za předpokladu, že dx a dy představují posunutí pixelu v čase dt, lze pravou část obrázku zapsat jako $I(x + dx, y + dy, t + dt)$. Z předpokladu konstantní intenzity pixelu vyplývá rovnice:

$$I(x,\ y, t) =\ I(x + dx, y + dy, t + dt) \tag{2.2}$$

Pravou stranu této rovnice lze rozložit na Taylorovu řadu v bodě (x, y, t):

$$I\left(x + dx, y + dy, t + dt\right) = I\left(x,\ y,\ t\right) + \frac{dI}{dx}dx + \frac{dI}{dy}dy + \frac{dI}{dt}dt \tag{2.3}$$

Levou a pravou stranu rovnice lze zkratit a celou rovnici vydělit dt:

$$I_x u + I_y v + I_t = 0 \tag{2.4}$$

kde: $I_x = \frac{dI}{dx}$; $I_y = \frac{dI}{dy}$; $I_t = \frac{dI}{dt}$; $u = \frac{dx}{dt}$; $v = \frac{dy}{dt}$

Tato rovnice se nazývá rovnice optického toku. Zde I_x, I_y jsou prostorové gradienty a I_t je časový gradient obrazu, jsou známy. Proměnné u, v jsou horizontální a vertikální rychlost pixelu, které zůstávají neznámé. Jednomu pixelu odpovídá jedna rovnice se dvěma neznámými. Problém nedostatečnosti rovnice optického toku se nazývá aperturní problém. Tento problém se řeší Lucasovou-Kanadovou metodou, která vychází z druhého předpokladu.

V metodě Lucas-Canade je obraz rozdělen na malé čtvercové oblasti, např. 3×3, tj. tato oblast obsahuje 9 pixelů se stejným pohybem. K řešení soustavy 9 rovnic se dvěma neznámými se používá metoda nejmenších čtverců. Pro usnadnění zapíšeme soustavu rovnic v maticovém tvaru:

$$A \cdot b = c \tag{2.5}$$

kde: $A = \begin{pmatrix} I_{x1} & I_{y1} \\ I_{x2} & I_{y2} \\ \vdots & \vdots \\ I_{x9} & I_{y9} \end{pmatrix}$; $b = \begin{pmatrix} u \\ v \end{pmatrix}$; $c = \begin{pmatrix} I_{t1} \\ I_{t2} \\ \vdots \\ I_{t9} \end{pmatrix}$;

Matice A obsahuje prostorové gradienty obrazu a vektor c představuje časové změny intenzity pixelů. Řešení metodou nejmenších čtverců poskytuje vektor rychlosti b popisující pohyb pixelů:

$$b = \left(A^T A\right)^{-1} A^T c \tag{2.6}$$

Po zkraceni výrazu získáme konečný vzorec:

$$\begin{pmatrix} u \\ v \end{pmatrix} = \begin{pmatrix} \sum_i I_{x_i}^2 & \sum_i I_{x_i} I_{y_i} \\ \sum_i I_{x_i} I_{y_i} & \sum_i I_{y_i}^2 \end{pmatrix}^{-1} \begin{pmatrix} -\sum_i I_{x_i} I_{t_i} \\ -\sum_i I_{y_i} I_{t_i} \end{pmatrix} \tag{2.7}$$

Výsledkem metody je sada vektorů popisujících polohu a orientaci stejných bodů na po sobě jdoucích snímcích:

$$V_t = (x_i, \quad y_i, \quad u_i, \quad v_i)_{i=1}^{N} \tag{2.8}$$

kde:

- N - počet sledovaných klíčových bodů,
- x_i, y_i - souřadnice i-tého klíčového bodu ve snímku t,
- u_i, v_i - posun i-tého klíčového bodu vodorovně a svisle ze snímku t do snímku $t + 1$.

Optický tok lze rozdělit na **řídký (sparse)** a **hustý (dense)**, v závislosti na počtu sledovaných bodů. Metoda Lucase-Kanade vypočítává optický tok pro řídkou sadu klíčových bodů, které jsou předem vybírány pomocí detektoru rohů Harrise nebo Shi-Tomasi. Na rozdíl od toho metoda Farneback (Farneback Optical Flow) umožňuje vypočítat hustý optický tok pro každý pixel obrazu. Vysoká výpočetní složitost metody Farneback však činí tuto metodu méně vhodnou pro úlohy v reálném čase.

Metoda Lucase-Kanade je široce využívána díky své jednoduchosti a efektivitě. Je však třeba vzít v úvahu, že tato metoda je citlivá na šum a náhlé pohyby a změny jasu.

2.1.2 Blokové porovnávání s dekompozicí na bitové vrstvy

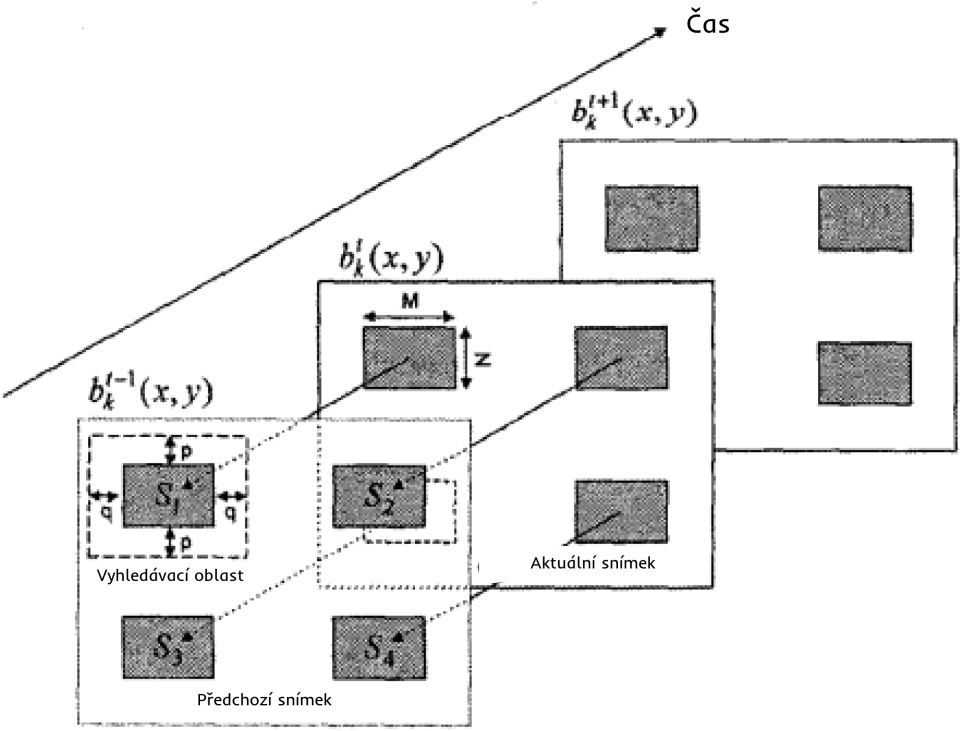
Algoritmus blokového porovnávání (Block-Matching Algorithm, BMA) [27] je jednou z nejjednodušších metod pro odhad pohybu ve videu. Blokové porovnávání se široce využívá ve video kodecích pro efektivní kompresi dat, stejně jako v systémech stabilizace obrazu a video dohledových systémů.

BMA předpokládá vysokou korelaci mezi každým pixelem a jeho sousedy. Princip blokového porovnávání spočívá v rozdělení aktuálního snímku na obdélníkové bloky S o velikosti $M \times N$ pixelů, pro každý blok se vytvoří oblast hledání v předchozím snímku, která má rozměry: $(2q + M) \times (2p + N)$ kde q a p jsou maximální možné posuny v horizontálním a vertikálním směru.

Lokální pohybové vektory se odhadují na základě čtyř podobrazů (S_I, S_2, S_3, S_4), které jsou umístěny na specifických pozicích v snímku. Pro každý podobraz v aktuálním snímku je pohybový vektor určen porovnáním s podobrazem v předchozím snímku. Jako výsledek je vybrán podobraz s nejlepším shodným výsledkem. Tento přístup předpokládá, že všechny pixely uvnitř podobrazu vykazují jednotný pohyb a rozsah pohybového vektoru je omezen vyhledávacím oknem. Na obrázku 2.2 je znázorněn proces blokového porovnávání [5].

Jako kritérium pro hodnocení míry shody mezi bloky se díky své jednoduchosti a nízkým výpočetním nákladům často používá součet absolutních rozdílů SAD (Sum of Absolute Differences) [28]. Pro každý pár pixelů ze srovnávaných bloků se vypočítá absolutní rozdíl hodnot jejich jasu, a tyto rozdíly se následně sečtou, aby se získala konečná hodnota SAD. Tento proces se opakuje pro všechna potenciální umístění bloku v rámci vyhledávacího okna. Blok s minimální hodnotou SAD je považován za nejlepší shodu. Chyba shody mezi blokem na pozici (x, y) v aktuálním snímku I_t a porovnávaným blokem na pozici $(x + u, y + v)$ v předchozím snímku I_{t-1} je definována následujícím vzorcem:

$$SAD = \sum_{i=1}^{M} \sum_{j=1}^{N} |I_t\,(x+i,\ y+j) - I_{t-1}\,(x+u+i,\ y+v+j)\,| \tag{2.9}$$



Obr. 2.2: Algoritmus blokového porovnávání [5]

kde: M a N představují šířku a výšku bloku.

Jednou z optimalizací metody BMA spočívá v dekompozici obrazu na bitové vrstvy [5, 27]. V horních významných bitových vrstvách, které obsahují hlavní informace o pohybu, může být algoritmus blokového porovnávání aplikován efektivněji. V tomto případě je jako kritérium hodnocení podobnosti mezi blokem na pozici (x, y) v aktuálním snímku I_t a porovnávaným blokem na pozici $(x+u, y+v)$ v předchozím snímku I_{t-1} definována následující formule:

$$SAD = \sum_{i=1}^{M}\sum_{j=1}^{N} {I^k}_t\,(x+i,\ y+j) \oplus {I^k}_{t-1}\,(x+u+i,\ y+v+j) \tag{2.10}$$

kde:

- ${I^k}_t\,(x+i,\ y+j)$ - k-tá bitová vrstva pixelu $(x+i, y+j)$ ve snímku t,
- ${I^k}_{t-1}\,(x+u+i, y+v+j)$ - k-tá bitová vrstva pixelu $(x+u+i, y+v+j)$ ve snímku $t-1$,
- $\oplus$ - bitový XOR.

Výběr velikosti bloku a oblasti hledání je kompromisem mezi přesností a výpočetní náročností. Malé bloky lépe zachycují drobné pohyby, ale jsou náchylné k šumu, zatímco velké bloky jsou méně přesné, ale odolné vůči šumu.

Blokové porovnávání vrací sadu vektorů, které popisují posun obdélníkových bloků mezi po sobě jdoucími snímky:

$$V_t = (x_i, \quad y_i, \quad u_i, \quad v_i)_{i=1}^{N} \tag{2.11}$$

kde:

- N - počet bloků, na které bylo rozděleno zobrazení ve snímku t,
- x_i, y_i - souřadnice levého horního pixelu i-tého bloku ve snímku t,
- u_i, v_i - posun i-tého bloku vodorovně a svisle ze snímku $t-1$ do snímku t.

Hlavním omezením algoritmu blokového porovnávání je předpoklad, že bloky se pohybují homogenně, se stejným vektorem. To vede k chybám v oblastech s nehomogenním pohybem nebo při přítomnosti rotací a deformací.

2.1.3 Porovnávání klíčových bodů s detektorem ORB

Porovnávání klíčových bodů je jednou z klasických metod pro odhad pohybu v počítačovém vidění, zejména při stabilizaci videa a v systémech simultánní lokalizace a mapování (SLAM). Obecně metoda spočívá v detekci a porovnávání významných oblastí obrazu, které zůstávají stabilní při změně úhlu pohledu, osvětlení a škálování [22, 29].

Detektor ORB identifikuje sady klíčových bodů ve dvou po sobě jdoucích snímcích $t-1$ a t. Každý klíčový bod uchovává souřadnici a oktávu, a také má deskriptor, který je porovnáván pomocí Hammingovy vzdálenosti. Pro každý klíčový bod ve snímku $t-1$ se pomocí algoritmu Brute Force Matcher nalezne odpovídající klíčový bod ve snímku t. Souřadnice spárovaných bodů mezi dvěma snímky umožňují odhad transformace mezi snímky.

Výsledkem je sada spojených párů klíčových bodů mezi předchozím a aktuálním snímkem, přičemž body bez odpovídajícího páru jsou vyřazeny.

$$V_{t-1} = (x_i, y_i, x_i', y_i')_{i=1}^{N} \tag{2.12}$$

kde:

- N – počet párů klíčových bodů mezi snímky $t-1$ a t,
- x_i, y_i – souřadnice i-tého klíčového bodu ve snímku $t-1$, x_i', y_i' – souřadnice i-tého klíčového bodu ve snímku t.

2.2 Fáze kompenzace pohybu

Během fáze odhadu pohybu se vytvářejí globální vektory, které určují trajektorii kamery. Tato trajektorie však často obsahuje ostré a nežádoucí výkyvy způsobené náhodnými pohyby kamery.

Cílem fáze vyhlazování trajektorie je oddělit záměrné pohyby kamery od nezáměrných prudkých pohybů a výkyvů, které vznikají během natáčení. Výsledkem je hladká trajektorie, stabilní a vizuálně příjemný obraz, přičemž přirozenost pohybu je zachována [1].

Stejně jako ve fázi odhadu pohybu existuje mnoho přístupů k dosažení hladké trajektorie. V této kapitole bude zvážena statistická metoda – Kalmanův filtr. Tento algoritmus efektivně zpracovává šumová data, má nízkou výpočetní náročnost a jeho rekurzivní povaha umožňuje jeho použití v zařízeních pracujících v reálném čase. Kalmanův filtr si dobře poradí jak s ostrými, tak s plynulými změnami trajektorie, a jeho široké rozšíření a dostupnost hotových knihoven usnadňují integraci do reálných systémů pro stabilizaci videa.

V oblasti stabilizace videa existuje mnoho alternativ ke Kalmanovu filtru. V literatuře se často objevuje třída jednoduchých filtrů (klouzavý průměr, exponenciální filtr). Tyto metody neberou v úvahu dynamiku systému a nemohou efektivně předpovídat trajektorii; v podmínkách vysokofrekvenčního šumu mohou buď příliš agresivně vyhlazovat, nebo naopak propouštět významné výkyvy. Je to způsobeno tím, že tyto filtry fungují na základě okna předchozích hodnot.

Dalšími metodami, jako je polynomiální aproximace nebo spline interpolace, nejsou optimální volbou pro tuto práci. Tyto metody vyžadují předem známou trajektorii a lépe se hodí pro offline zpracování, ale špatně fungují v podmínkách dynamicky se měnících scén a mají vysoké požadavky na výpočetní zdroje.

Nevhodná je také L1-optimalizace navržená Google Research. Ačkoli tato metoda poskytuje vysokou kvalitu stabilizace, vyžaduje značné výpočetní prostředky pro nalezení globálního minima, což ji činí nevhodnou pro zařízení s nízkým výkonem.

2.2.1 Kalmanův Filtr

Kalmanův filtr [30] je matematický nástroj, který zajišťuje rekurzivní odhad stavu systému s cílem minimalizovat střední kvadratickou chybu při určitých předpokladech, jako je linearita systému a gaussovský šum. Umožňuje odhadovat aktuální a předpovídat budoucí stav systému, i když přesná povaha modelovaného systému není známa.

Filtr byl pojmenován po svém autorovi, Rudolfu Kalmanovi, který tuto metodu vyvinul v roce 1960. Od té doby se metoda stala standardem pro úlohy sledování a předpovídání stavů v oblastech, jako je teorie řízení, zpracování signálů a digitální stabilizace videa.

Kalmanův filtr je lineární rekurzivní metoda a je založen na Markovově vlastnosti, podle které aktuální stav procesu závisí pouze na předchozím stavu procesu,

nikoli na celé sérii minulých stavů. Markovská povaha filtru jej činí vhodným pro úkol stabilizace videa v reálném čase [31].

Uvažme lineární dynamický systém bez vstupů v diskrétním čase, který je vystaven bílému šumu [32]. Systém je popsán následujícím modelem:

$$x_k = A_{k-1}x_{k-1} + G_{k-1}w_{k-1} \tag{2.13}$$

$$z_k = H_k x_k + v_k \tag{2.14}$$

Rovnice (2.13) se nazývá stavová rovnice, která popisuje dynamiku systému, kde:

- $x_k \in \mathbb{R}^n$ - stavový vektor systému v časovém okamžiku t_k,
- $A_{k-1} \in \mathbb{R}^{nxn}$ - stavová přechodová matice mezi časovými okamžiky t_{k-1} a t_k,
- w_{k-1} - náhodný šum procesu s kovarianční maticí $Q_{k-1} \in \mathbb{R}^{lxl}$,
- $G_{k-1} \in \mathbb{R}^{nxl}$ - matice vlivu šumu na následující stav systému.

Rovnice (2.14) se nazývá měřicí rovnice, kde:

- $z_k \in \mathbb{R}^m$ - vektor měření v časovém okamžiku t_k,
- $H_k \in \mathbb{R}^{mxn}$ - pozorovací matice (spojuje stav systému s měřeními),
- v_k - náhodný šum měření s kovarianční maticí $R_{k-1} \in \mathbb{R}^{mxm}$.

Kovarianční matice [30] je čtvercová matice, jejíž prvky představují kovariance mezi páry náhodných veličin z datového vektoru. Popisuje rozptyl hodnot náhodných veličin a závislosti mezi nimi. Matice šumu procesu Q a matice šumu měření R popisují nejistotu v dynamice modelu a měřeních. Matice chyby odhadu P vyjadřuje míru jistoty odhadu stavu.

Algoritmus je založen na následujících předpokladech [32]:

- Šumy procesu a měření mají normální rozdělení a nejsou vzájemně korelovány.
- Střední hodnota šumů je nulová, tedy nemá systematické zkreslení.
- Počáteční stav systému x_0 a počáteční kovarianční matice P_0 jsou známy.
- Matice $A,\ H,\ Q,\ R,\ G$ jsou známy.

Kalmanův filtr odhaduje proces pomocí řízení se zpětnou vazbou [30]. Filtr a priori odhaduje stav procesu v časovém okamžiku t_k a poté koriguje a priori odhad na základě zpětné vazby ve formě šumových měření, kde korigovaný odhad stavu se nazývá a posteriori. Algoritmus filtru lze rozdělit do dvou kroků: Predikce (Time Update) a Korekce (Measurement Update).

Krok predikce, jak napovídá název, hledí vpřed v čase, aby a priori odhadl následující stav systému a kovarianci chyby na základě aktuálního stavu. Předpověď stavu:

$$x'_k = A_{k-1}x_{k-1} \tag{2.15}$$

kde:

- x'_k - apriorní hodnocení stavu v kroku k,
- x_{k-1} - aposteriorní hodnocení stavu v kroku $k-1$.

Předpověď kovarianci chyby:

$$P'_k = A_{k-1}P_{k-1}A_{k-1}^T + G_{k-1}Q_{k-1}G_{k-1}^T \tag{2.16}$$

kde:

- P'_k - apriorní kovariační matice stavové chyby v kroku k,
- P_{k-1} - aposteriorní kovariační matice stavové chyby v kroku $k-1$.

Krok korekce porovnává apriorní hodnocení stavu se skutečnými měřeními, aby se dosáhlo zlepšeného aposteriorního hodnocení.

Aposteriorní hodnocení stavu a kovariací chyby:

$$x_k = x'_k + K_k\left[z_k - H_k x'_k\right] \tag{2.17}$$

$$P_k = P'_k - K_k H_k P'_k \tag{2.18}$$

$$K_k = P'_k H_k^T \left[H_k P'_k H_k^T + R_k\right]^{-1} \tag{2.19}$$

kde:

- x_k - aposteriorní hodnocení stavu v kroku k,
- P_k - aposteriorní kovariační matice chyby stavu,
- K_k - zesílení Kalmánu.

Po každém páru predikce a korekce se proces opakuje s předchozí a posteriori hodnotou odhadu, která se používá pro předpověď nových a priori odhadů [30].

Ve stabilizaci videa [8] je poloha centrálního bodu prvního snímku nastavena jako počátek souřadnicového systému. Poloha centrálního bodu každého snímku bez chvění je určena souřadnicemi (x_k, y_k), pozorovaná hodnota těchto souřadnic je označena jako (x_{ok}, y_{ok}), (dx, dy) popisuje rychlost změny centrálního bodu.

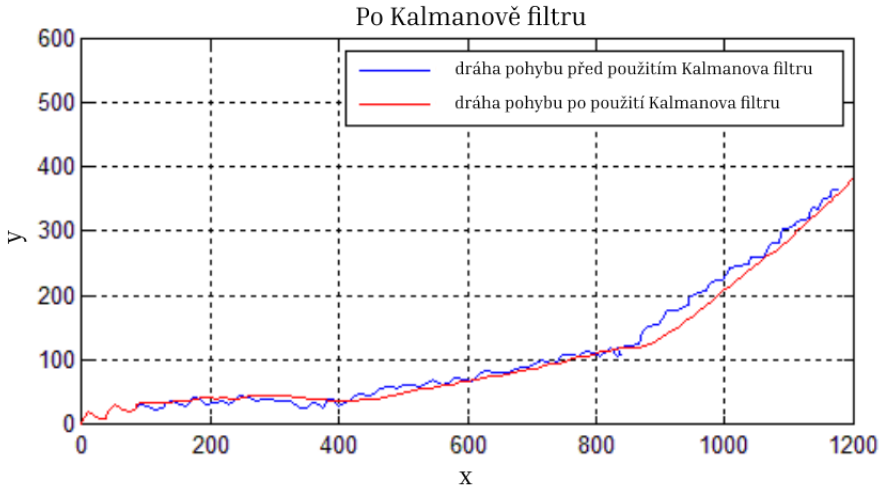
$$\begin{pmatrix} x_{k+1} \\ y_{k+1} \\ dx \\ dy \end{pmatrix} = \begin{pmatrix} 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x_k \\ y_k \\ dx \\ dy \end{pmatrix} + W_k \tag{2.20}$$

$$\begin{pmatrix} x_{ok} \\ y_{ok} \\ dx \\ dy \end{pmatrix} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x_k \\ y_k \\ dx \\ dy \end{pmatrix} + V_k \tag{2.21}$$

Rovnice (2.20) a (2.21) představují zjednodušený zápis stavové a měřicí rovnice pro systém stabilizace videa. Výsledkem je stavová přechodová matice A a pozorovací

matice H. Kovarianční matice šumu systému Q a kovarianční matice šumu měření R jsou nastaveny na dostatečně malé hodnoty; zdroj [8] doporučuje hodnotu 10^{-5}. Počáteční hodnoty x_0 a P_0 jsou nastaveny na náhodné hodnoty.

Demonstrace Kalmanova filtru aplikovaného na stabilizaci videa je znázorněna na obrázku 2.3



Obr. 2.3: Efekt Kalmanova filtru pro nestabilní trajektorii kamery [8]

2.3 Fáze deformace obrazu

V procesu stabilizace videa je hlavním cílem odstranit nežádoucí zkreslení způsobená neúmyslnými pohyby kamery. Taková zkreslení se mohou projevovat ve formě chvění obrazu, náhodných posunů a otáčení záběru.

Ve fázi odhadu pohybu se vypočítávají výchozí afinní transformace T_t, které popisují mezisnímkový pohyb kamery. V těchto transformacích se odráží nejen skutečný pohyb (například panorámování či náklon), ale i šumová složka způsobená chvěním kamery.

Dále se v etapě vyhlazování trajektorií používá Kalmanův filtr, který redukuje vysokofrekvenční šum. V důsledku této úpravy se vytvářejí nové parametry afinní transformace. Předpokládá se, že matice T_t' odpovídá „úmyslnému“ (tedy požadovanému) pohybu bez náhlých výkyvů. Rozdíl mezi T_t' a T_t odráží nežádoucí transformace, které je třeba kompenzovat.

Proces kompenzace [33] spočívá v použití „inverzní“ výchozí transformace a současném aplikování úmyslné transformace. Matematicky je to vyjádřeno následujícím způsobem:

$$x_t{}' = T_t{}^{-1} T_t' x_t \tag{2.22}$$

kde:

- x'_t - nové souřadnice pixelu po stabilizaci snímku t,
- x_t - původní souřadnice pixelu v snímku t,
- T_t^{-1} - inverzní matice původní afinní transformace,
- T_t - vyhlazená“ (úmyslná) afinní transformační matice.

Tento vzorec se aplikuje na každý pixel původního snímku, aby se nakonec vytvořil kompenzovaný snímek. Avšak souřadnice x'_t se v obecném případě neshodují s celočíselnými hodnotami odpovídajícími pozicím diskrétní sítě pixelů výsledného obrazu. To vede k nutnosti provádět interpolaci, protože při jednoduchém zaokrouhlení souřadnic mohou vznikat „roztříštěné“ hrany a černé oblasti, kde data chybí nebo jich není dostatek.

3 Implementace zvoleného algotitmu

3.1 Úvod do řešení

Tato kapitola popisuje praktickou implementaci algoritmu stabilizace videa v reálném čase, který byl zvolen v rámci semestrální práce. Algoritmus je založen na odhadu pohybu pomocí porovnání klíčových bodů s využitím detektoru ORB, filtrování chybných shod metodou RANSAC a vyhlazování trajektorie pohybu pomocí Kalmanova filtru, předpokládá afinní 2D modelu pohybu kamery. Tento model umožňuje stabilizovat vertikální a horizontální posuny i rotaci kolem osy Z v rovině obrazu.

Vyvinuté softwarové řešení umožňuje efektivně vyhlazovat mírné chvění ve videostreamu — jak při zpracování uložených videozáznamů, tak i při práci s živým streamem přímo z kamery. Architektura systému je navržena podle principů modularity a objektově orientovaného programování, což zajišťuje snadnou údržbu a rozšiřitelnost kódu. Kromě hlavního modulu stabilizace byly implementovány také pomocné komponenty, které poskytují API pro flexibilní nastavení parametrů algoritmu dle podmínek scény a pro řízení provozních režimů.

Navržená architektura také umožňuje škálování a nahrazení jednotlivých komponent algoritmu. Například v případě potřeby může být Kalmanův filtr nahrazen klouzavým průměrem nebo rozšířeným Kalmanovým filtrem v závislosti na požadavcích na výkon.

Vývoj a testování softwarového řešení probíhalo na dvou platformách: notebooku s operačním systémem Windows a jednodeskovém počítači Raspberry Pi 4B se systémem Linux. Implementovaný algoritmus tak zajišťuje multiplatformní kompatibilitu pro uvedené systémy.

Tato kapitola dále podrobně popisuje architekturu systému, strukturu zdrojového kódu, funkce jednotlivých modulů, možnosti konfigurace programu a poskytuje praktický návod pro jeho rychlé spuštění.

3.2 Architektura programu

Softwarové řešení je implementováno v jazyce Python 3 a navrženo v souladu se zásadami objektově orientovaného programování, což zajišťuje modularitu, přehlednost a možnost rozšiřování. Architektura aplikace je rozdělena do specializovaných komponent, z nichž každá plní přesně definovanou úlohu v rámci celkového procesu stabilizace. Hlavním vstupním bodem programu je skript `run.py`, který obsahuje řídicí logiku běhu algoritmu.

Po spuštění programu se očekává přítomnost konfiguračního souboru `config.json` ve stejné složce jako spouštěcí skript. Pokud soubor není nalezen, bude automaticky vytvořen. Tento soubor může obsahovat buď uživatelsky definované parametry, nebo obsahovat prázdný slovník – v takovém případě se použijí výchozí hodnoty. Načtení a validace konfigurace probíhá prostřednictvím funkce `load_and_validate_config()`, která ověřuje správnost datových typů a hodnotových rozsahů. Pokud některé parametry chybí, jsou doplněny výchozími hodnotami; při nesprávných hodnotách je vyvolána výjimka s popisem chyby.

Na základě načtené konfigurace jsou inicializovány základní komponenty systému. Třída `FrameSource` odpovídá za připojení ke zdroji snímků — tím může být buď lokální videosoubor, nebo živý přenos z kamery. Podle konfigurace aktivují moduly vizualizace trajektorie (`PlotTrajectory`) a logování (`Logger`), které umožňují sledovat chování algoritmu v průběhu zpracování a volitelně ukládat ladicí informace.

Klíčovým prvkem systému je objekt třídy `Stabilizer`, který implementuje hlavní algoritmus digitální stabilizace. Na vstupu přijímá první snímek a konfiguraci. Pokud je aktivní režim ukládání výstupu, je inicializována komponenta `VideoWriter`.

Hlavní zpracovatelská smyčka je koncipována jako opakované zpracování jednotlivých snímků. V každé iteraci je nejprve načten aktuální snímek ze zdroje a předán modulu pro stabilizaci. Po získání stabilizované verze snímku jsou následně prováděny další činnosti, které jsou podmíněny konfiguračním nastavením zadaným při inicializaci systému:

1. Oříznutí okrajů stabilizovaného snímku dle zadaných parametrů.
2. Stabilizovaný snímek může být na základě nastavení převeden do kombinované podoby, která spojuje původní a stabilizovaný obraz vedle sebe pro účely srovnání.
3. Zobrazení i ukládání výstupního snímku jsou řízeny konfiguračním nastavením. Stabilizovaný snímek může být zobrazen nebo uložen buď samostatně, nebo ve formě kombinace s původním obrazem.
4. Aktualizace provozních statistik (počet úspěšně zpracovaných snímků, FPS, využití CPU, paměti).
5. Sběr trajektorických dat.

Zpracování pokračuje, dokud jsou k dispozici vstupní snímky, nebo dokud není program ručně ukončen uživatelem stiskem klávesy Esc.

Vizualizaci je možné zcela vypnout pro zvýšení výkonu.

3.3 Implementační struktura a komponenty programu

V této podkapitole jsou podrobně analyzovány klíčové moduly softwarového systému: jejich účel, struktura, hlavní třídy a metody, stejně jako kontext, ve kterém

jsou používány. Pro každý modul je uveden popis jeho role v celkovém stabilizačním procesu, způsob interakce s ostatními komponentami a případné specifikum závislé na použité platformě.

3.3.1 Modul `source.py` — třída `FrameSource`

Účel a využití: Třída `FrameSource` představuje abstrakci zdroje videostreamu. Poskytuje jednotné rozhraní pro získávání snímků jak z lokálního videosouboru, tak přímo z kamery. Je využívána v hlavním modulu `run.py` pro sekvenční čtení snímků během běhu hlavní logiky aplikace.

Interní logika a komponenty: Instance třídy je inicializována pomocí konfiguračního slovníku, který určuje typ a parametry vstupního zdroje. Pro zajištění multiplatformní kompatibility využívá třída odlišné implementace v závislosti na provozní platformě:

- Na platformě Raspberry Pi je použitá knihovna `Picamera2`.
- Na systému Windows je využit objekt `cv.VideoCapture` z knihovny OpenCV.

Hlavní metody:

- `read()` - Vrací aktuální snímek jako `NumPy` pole. Pokud čtení selže, vrací `None`.
- `release()` - Korektně uvolní prostředky spojené se vstupním streamem. Tato metoda by měla být zavolána před ukončením programu.

3.3.2 Modul `visualizer.py` — třída `TrajectoryPlotter`

Účel a využití: Třída `TrajectoryPlotter` je určena ke sběru a vizualizaci odhadované a stabilizované trajektorie pohybu kamery. Používá se v hlavním modulu `run.py` a plní podpůrnou roli v procesu stabilizace tím, že poskytuje nástroje pro analýzu účinnosti algoritmu.

Interní logika a komponenty: Instance třídy je při inicializaci vytvořena s konfiguračním slovníkem. Pokud je v konfiguraci aktivován příznak vizualizace trajektorie, jsou vytvořeny vnitřní datové struktury pro akumulaci hodnot. Výstupem je grafické zobrazení surové a vyhlazené trajektorie. Pokud není vizualizace aktivována, třída neprovádí žádné operace. Díky tomu je bezpečné objekt volat i v případech, kdy uživatel nechce vizualizaci zobrazovat, bez vlivu na funkčnost systému.

Hlavní metody:

- `collect(trajectory)` - Volána v každé iteraci hlavní smyčky a získává data prostřednictvím ústřední metody třídy `Stabilizer`. Ukládá odhadnuté a stabilizované hodnoty pohybu do příslušných polí.
- `display()` - Používá se po ukončení zpracování. Vykresluje grafy znázorňující trajektorii kamery před a po stabilizaci.

3.3.3 Modul `logger.py` — třída `Logger`

Účel a využití: Třída `Logger` je určena pro výpis diagnostických zpráv během běhu algoritmu stabilizace, včetně varování při nevalidním odhadu pohybu. Dále zajišťuje sběr a zobrazení souhrnné statistiky každých 100 zpracovaných snímků. Jedná se o podpůrný modul, který slouží k monitorování výkonnosti a stability celého systému.

Objekt `Logger` je vytvořen v hlavním souboru `run.py`, předáván do `Stabilizer` a následně do modulu `MotionEstimator`. Využívá se zejména k zaznamenávání událostí, jako jsou nedostatek odpovídajících bodů, statická scéna či porucha vstupních dat.

Interní logika a komponenty: `Logger` je inicializován pomocí konfiguračního slovníku. Pokud je v konfiguraci aktivní příznak pro logování nebo měření výpočetní zátěže, komponenta se aktivuje; v opačném případě nevykonává žádné operace. V případě, že je zadána výstupní cesta k souboru, log je akumulován v paměti pro pozdější uložení. V opačném případě je výstup omezen na konzoli. Pokud je zapnut režim měření výpočetní zátěže, každých 100 snímků se vypočítává průměrné FPS, využití CPU, paměti a podíl neúspěšných iterací. Tato diagnostická informace je pak zaznamenána v rámci logu.

Hlavní metody:

- `log(msg, type="INFO")` - Zapíše zprávu zvoleného typu. Pokud je aktivní historie, přidá ji do paměti pro pozdější uložení.
- `update_status(success)` - Funkce se volá během každé iterace stabilizace snímků ve `stabilizer/__init__.py`. Měří výpočetní zátěž a zaznamenává statistiku neúspěšně stabilizovaných snímků. Každých 100 snímků vypisuje souhrnnou diagnostiku.
- `save_log()`: Uloží akumulovaný výstup do souboru, pokud tato možnost byla povolena v konfiguraci. Používá se po ukončení zpracování.

3.3.4 Modul `stabilizer/__init__.py` — třída `Stabilizer`

Účel a využití: Třída `Stabilizer` implementuje vysokou úroveň logiky digitální stabilizace videa. Spojuje tři klíčové komponenty: odhad pohybu mezi snímky, vyhlazení trajektorie a aplikaci korekční transformace. Jedná se o ústřední prvek celého procesu stabilizace, který koordinuje zpracování jednotlivých snímků.

Objekt `Stabilizer` je vytvářen v hlavním modulu `run.py`, kde přijímá první vstupní snímek, objekt `Logger` a konfigurační slovník s parametry, které jsou dále předány k vytvoření vnitřních komponent. Slouží jako prostředník mezi zpracováním pohybu a transformací výsledného snímku.

Interní logika a komponenty: Při inicializaci jsou vytvořeny následující komponenty:

- Objekt `MotionEstimator` – Zajišťuje výpočet odhadu pohybu mezi aktuálním a předchozím snímkem.
- Objekt `MotionFilter` – Zkumuluje a vyhlazuje sekvenci pohybů, následně určuje korekci.
- Objekt `Logger` - Volitelné logování informací o scéně, odhadu pohybu, statistiky zatížení apod.
- NumPy pole `last_stable` - Poslední stabilizovaný snímek

Hlavní metody:

- `stabilize(curr)` - Hlavní metoda pro zpracování snímku. Přijímá aktuální snímek a objekt logování.

  Postup:

  1. Odhad pohybu mezi aktuálním a předchozím snímkem.
  2. Vyhlazení trajektorie pomocí `MotionFilter` a výpočet korekce.
  3. Aplikace korekce pomocí funkce `warp_frame(...)`.
  4. Při selhání odhadu pohybu vrátí poslední stabilizovaný snímek.
  5. Případná aktualizace loggeru a výpis diagnostiky.

  Vrací výsledný snímek jako NumPy pole.
- `export_trajectory_data()`: Vrací surové a vyhlazené hodnoty trajektorie pro účely vizualizace.

3.3.5 Modul `stabilizer/estimator.py` — třída `MotionEstimator`

Účel a využití: Třída `MotionEstimator` zajišťuje výpočet odhadu mezisnímkového pohybu pomocí párování klíčových bodů. Výsledný vektor pohybu (`dx`, `dy`, `dr`) slouží třídě `Stabilizer` pro konstrukci trajektorie a korekci obrazu.

Interní logika a komponenty: Při vytvoření přijímá první snímek a konfigurační parametry. Interně vytváří:

- ORB detektor,
- Brute Force Matcher s Hammingovou metrikou,
- předchozí snímek jako objekt třídy `FrameFeatures`,
- kruhový buffer středních rozdílů jasnosti o definované délce (pro detekci statické scény),

Hlavní metody:

- `is_static_scene(curr)` — Přijímá aktuální snímek jako `FrameFeatures`, spočítá průměrný absolutní rozdíl mezi aktuálním a předchozím snímkem (v odstínech šedi), uloží výsledek do bufferu a vrací `True`, pokud je průměr z posledních 10 hodnot pod definovaným prahem. Používá se jako mechanismus

necitlivosti na mikropohyby a šum, čímž se předchází falešné aktivaci stabilizace.

- `estimate(curr_frame, logger)` — Přijímá aktuální snímek (jako NumPy pole) a objekt třídy `Logger`. Vrací vektor pohybu. Postup:
  1. Vyhodnocení statické scény, zaznamenání změn stavu (statická/dynamická scéna) a do logu.
  2. Detekce a párování bodů pomocí ORB a matcheru.
  3. Výpočet afinní transformace pomocí RANSAC.
  4. Extrakce translace a rotace.
  5. Návrat `None` při selhání.
  6. V případě nevalidní estimace je generováno odpovídající upozornění.
  7. Důležitou součástí výpočtu je skutečnost, že odhad probíhá nad zmenšeným snímkem, zatímco výsledná transformace se vztahuje ke snímku v původní velikosti. Proto se posuny `dx` a `dy` škálují zpět dle `resize_ratio`.

Je důležité poznamenat, že vypočtený pohyb nemusí nutně odpovídat reálnému pohybu kamery. Například pohybující se objekty v jinak statické scéně mohou způsobit falešnou detekci pohybu celé scény.

3.3.6 Modul `stabilizer/frame_features.py` — třída `FrameFeatures`

Účel a využití: Třída `FrameFeatures` slouží jako kontejner pro snímek. Každý snímek je v rámci odhadu pohybu reprezentován instancí této třídy, což zajišťuje, že detekce ORB a převod do odstínů šedi jsou provedeny pouze jednou pro daný snímek. Používá se v modulu `MotionEstimator` ke snížení výpočetní náročnosti.

Interní logika a komponenty: Uchovává původní barevný snímek, jeho zmenšenou verzi v odstínech šedi, detekované klíčové body a odpovídající deskriptory, které jsou vypočítány na základě této zmenšené verze.

Menší vstupní obraz výrazně zrychluje detekci klíčových bodů a tím i celý proces stabilizace. Příliš velké zmenšení však může negativně ovlivnit přesnost výpočtu pohybu, protože v obraze bude méně informací pro spolehlivé párování.

Při vytvoření třídy provede následující kroky:

- Zmenšení vstupního snímku podle parametru `resize_ratio`.
- Převod zmenšeného snímku do odstínů šedi.
- Detekce klíčových bodů a výpočet deskriptorů pomocí předaného ORB detektoru.

3.3.7 Modul `stabilizer/smoother.py` — třída `MotionFilter` a `Kalman1D`

Účel a využití: Třída `MotionFilter` slouží k akumulaci odhadovaného mezisnímkového pohybu do globální trajektorie kamery a k jejímu následnému vyhlazení. Každá složka vektoru pohybu (vodorovný posun, svislý posun, rotace) je zpracována nezávislým filtrem typu `Kalman1D`. Třída je využívána ve třídě `Stabilizer` ke kumulaci trajektorie, výpočtu vyhlazené verze a určení korekčního posunu, který je aplikován na aktuální snímek.

Interní logika a komponenty:

- Třída uchovává kumulativní součty surové a vyhlazené trajektorie.
- Pro každou složku používá samostatnou instanci třídy `Kalman1D`, která implementuje jednoduchý jednorozměrný Kalmanův filtr s konstantními parametry `Q` a `R`.

Hlavní metody:

- `cumulate(raw_motion)` — Přičte nový vektor pohybu k dosavadní surové trajektorii.
- `smooth()` — Aplikuje Kalmanovy filtry na surovou trajektorii a aktualizuje její vyhlazenou podobu.
- `compute_correction()` — Vrátí korekční posun.
- `get_raw_smoothed_trajectory()` — Vrací šesticový vektor aktuální trajektorie. Prostřednictvím metody `export_trajectory_data()` třídy `Stabilizer` jsou informace o aktuálním stavu trajektorie předávány modulu `run.py` za účelem vykreslení trajektorie.

`Kalman1D` je pomocná třída, implementuje základní prediktivně-korekční schéma Kalmanova filtru pro jednorozměrný signál. Je používána třídou `MotionFilter` pro každou složku pohybu zvlášť. Parametry `Q` a `R` jsou definovány v konfiguraci a určují míru důvěry v predikci a měření.

3.3.8 Modul `stabilizer/transform.py`

Tento modul obsahuje jedinou, avšak klíčovou funkci `warp_frame(...)`, která realizuje korekční transformaci obrazu na základě odhadnutého pohybu.

Funkce `warp_frame(frame, corrective_motion)`: Přijímá snímek a vektor korekčního pohybu ve formátu translace `dx`, `dy` a úhel rotace `dr`. Na základě těchto hodnot vytvoří afinní transformační matici, která je následně aplikována na vstupní snímek pomocí funkce `cv2.warpAffine()`. Výsledkem je stabilizovaná verze obrazu.

Funkce je volána ve třídě `Stabilizer` po výpočtu korekčního posunu a představuje závěrečný krok stabilizace každého snímku.

3.3.9 Modul `utils.py`

Tento modul obsahuje obecné pomocné funkce, které jsou využívány napříč celým projektem. Jedná se zejména o validaci konfigurace, úpravu obrazu, výpis výsledků a správu výstupního souboru.

Funkce `load_and_validate_config(path)`: Načte konfigurační soubor typu JSON, ověří typy a rozsahy všech parametrů, a při nesprávné hodnotě vyvolá výjimku. Pokud je zdrojem videa soubor, je vyžadováno zadání jeho cesty; ostatní chybějící parametry jsou doplněny výchozími hodnotami. Funkce je volána na začátku programu v modulu `run.py`.

Funkce `show_result(config, frame_smooth, frame_raw)`: Vrací výstupní snímek — buď samostatně beze změn, nebo v kombinaci s původním snímkem, pokud je v konfiguraci aktivní příslušný příznak. Volitelně může také zobrazit výsledek na obrazovce. Funkce je volána v hlavním souboru `run.py`.

Funkce `crop_stabilized_frame(config, frame)`: V případě aktivního ořezávání okrajů (nastaveného v konfiguraci) vrací oříznutou verzi stabilizovaného snímku podle zadaných hodnot. V opačném případě vrací původní obraz. Používá se v hlavním modulu při závěrečném zpracování.

Funkce `init_video_writer(config, frame_size)`: Inicializuje objekt `cv2.VideoWriter` v závislosti na rozlišení, režimu ořezávání, kombinovaném zobrazení a aktivaci ukládání výsledku. Používá se v `run.py`.

Funkce `check_esc(is_display_on)`: Zajišťuje možnost přerušit běh programu pomocí klávesy Esc. Používá odlišné mechanismy pro platformy Windows a Linux a rozlišuje, zda je aktivní výstupní okno OpenCV.

3.4 Konfigurační soubor `config.json`

Konfigurační soubor `config.json` hraje klíčovou roli v architektuře programu, protože jeho obsah přímo ovlivňuje chování jednotlivých komponent i celé stabilizační pipeline. Parametrizace systému umožňuje přizpůsobit algoritmus konkrétním podmínkám scény a dostupným výpočetním prostředkům bez nutnosti zásahu do zdrojového kódu. Tento přístup zvyšuje flexibilitu systému a umožňuje uživateli experimentovat s různými nastaveními, případně získávat dodatečné ladicí informace v rámci testování.

3.4.1 Parametry vstupu

Parametry uvedené níže určují způsob získávání vstupních snímků, ať už ze souboru, nebo z kamery.

- `source_of_frames` - Určuje typ vstupního zdroje: `"camera"` nebo `"video"`. Ve výchozím nastavení v režimu `camera`.
- `input_video_path` - Cesta k videosouboru. Nemá výchozí nastavení a je vyžadována, pokud je jako vstupní zdroj zvoleno `"video"`.
- `picamera2_resolution` - Rozlišení pro kameru `Picamera2`. Ve výchozím nastavení `[640, 360]`.
- `picamera2_fps` - Vstupní FPS při použití `Picamera2`. Výchozí hodnota je `24`.

Parametry `picamera2_resolution` a `picamera2_fps` jsou relevantní pouze na platformě Raspberry Pi, pokud je jako zdroj snímků nastaven režim `"camera"`.

3.4.2 Režimy zobrazení

Následující parametry ovlivňují způsob vizuální prezentace výstupu programu, a to jak v reálném čase, tak při ukládání zpracovaných dat.

- `display_output` - Povolení zobrazení výstupních snímků během zpracování. Výchozí hodnota je `true`.
- `show_combined` - Zobrazuje původní a stabilizovaný snímek vedle sebe. Výchozí hodnota je `false`.
- `crop_result` - Aktivuje ořezání okrajů výstupního snímku. Výchozí hodnota je `false`.
- `margin_x` - Ořez zleva a zprava (v pixelech). Výchozí hodnota je `30`.
- `margin_y` - Ořez shora a zdola (v pixelech). Výchozí hodnota je `10`.
- `plot_trajectory` - Povolení vizualizace trajektorie pohybu kamery. Výchozí hodnota je `false`.

3.4.3 Parametry stabilizace

Tato skupina parametrů definuje chování hlavního stabilizačního algoritmu, včetně detekce pohybu, limitace transformace a nastavení filtru Kalman.

- `static_scene_threshold` - Práh pro detekci statické scény na základě rozdílu jasů. Při nule detekce statické scény vypnuta. Při příliš vysoké hodnotě se může stabilizace deaktivovat. Výchozí hodnota je `0`.
- `max_feature_count` - Maximální počet klíčových bodů ORB na snímku. Výchozí hodnota je `300`.
- `resize_ratio` — Škálovací faktor určuje míru zmenšení snímku před detekcí klíčových bodů; zrychluje zpracování, ale může snížit přesnost stabilizace. Výchozí hodnota je `1.0` (původní rozlišení).
- `kalman_Q` - Variance predikce v Kalmanově filtru. Výchozí hodnota je `1e-5`.

- `kalman_R` - Variance měření v Kalmanově filtru. Výchozí hodnota je `5e-2`.

3.4.4 Logování a ukládání

Pro účely ladění a archivace výsledků slouží následující volitelné parametry. Umožňují ukládat výstupní video a zaznamenávat interní stav během běhu algoritmu.

- `log_message` - Aktivuje logování zpráv o běhu stabilizace. Výchozí hodnota je `false`.
- `measure_performance` - Aktivuje měření a logování výpočetní zátěže během zpracování. Výchozí hodnota je `false`.
- `save_log_to` - Cesta k souboru, do kterého se uloží kompletní log. Pokud není zadána, výstup se neukládá. Výchozí hodnota je `null`.
- `save_output_video_to` - Cesta k výstupnímu videosouboru. Pokud není zadána, výstup se neukládá. Výchozí hodnota je `null`.
- `output_video_fps` - FPS ve výstupním videu. Výchozí hodnota je `25`.

3.5 Rychlý start

Tato část stručně popisuje, jak připravit a spustit program pro stabilizaci videa. Cílem je umožnit uživateli rychlé otestování systému bez nutnosti podrobného studia implementace.

3.5.1 Závislosti a prostředí

Pro úspěšné spuštění programu je nutné mít nainstalované následující knihovny:

- `opencv-python`
- `numpy`
- `matplotlib`
- `psutil`
- `picamera2` (pouze při použití na Raspberry Pi)

Doporučená verze Pythonu: `Python 3.8+`. Instalaci všech závislostí lze provést vložením tohoto příkazu do terminálu:

```
pip install opencv-python numpy matplotlib psutil
```

Na Raspberry Pi je nutné ještě doinstalovat závislosti pomocí tohoto příkazu:

```
pip install picamera2
```

3.5.2 Spuštění programu

1. Zajistěte, že ve složce s hlavním skriptem `run.py` se nachází konfigurační soubor `config.json`. Pokud soubor neexistuje, program se spustí s výchozími hodnotami.
2. Upravte parametry v `config.json` podle potřeby.
3. Spusťte program pomocí příkazu:

   ```
   python run.py
   ```

4. Výstup bude zobrazen na obrazovce, případně uložen dle konfigurace.

3.5.3 Ukázkový soubor `config.json`

```
{
    "source_of_frames": "video",
    "input_video_path": "vstup.mp4",
    "display_output": true,
    "show_combined": true,
    "save_output_video_to": "vystup.avi"
}
```

Tato konfigurace spustí stabilizaci z video souboru `vstup.mp4`, zobrazí kombinovaný výstup na obrazovce a zároveň uloží výsledek do souboru `vystup.avi`.

4 Vyhodnocení zvoleného algoritmu

4.1 Kvalitativní hodnocení

Tato kapitola se zaměřuje na hodnocení kvality digitální stabilizace videa, která byla navržena a implementována v rámci této práce.

Cílem je zjistit, do jaké míry algoritmus redukuje nežádoucí pohyby kamery a zlepšuje vizuální kvalitu záznamu. Hodnocení probíhá pomocí kvantitativních metrik založených na sledování klíčových bodů mezi snímky.

Pro objektivní posouzení kvality stabilizace byl zvolen přístup založený na analýze mezisnímkového posunu detekovaných klíčových bodů. Konkrétně se porovnávají vzdálenosti odpovídajících si bodů mezi sousedními snímky, které jsou detekovány pomocí algoritmu ORB a spárovány metodou Brute Force Matcher. Pro každý pár snímků se vypočítá průměrná eukleidovská vzdálenost mezi odpovídajícími body. Tato hodnota reprezentuje míru nestability daného úseku videa. Celkové hodnocení pak vychází z průměru a směrodatné odchylky těchto vzdáleností napříč celým videem. Nižší hodnoty značí plynulejší a stabilnější výstup.

Výsledky stabilizovaného videa jsou následně porovnány s původním (nestabilizovaným) záznamem. Pro zvýšení objektivity jsou navíc výsledky konfrontovány s alternativním algoritmem, který je volně dostupný v otevřeném softwarovém repozitáři [34].

Výpočet metrik byl proveden na testovacím videu `shakyTrain.mp4` s rozlišením `640:360` a snímkovou frekvencí 25 FPS. Toto video představuje typický případ neprofesionální ruční kamery. Během vývoje algoritmu byla jeho funkčnost testována právě na tomto videu, které dobře ilustruje slabé stránky implementovaného řešení. Tyto nedostatky jsou navíc typické pro všechny algoritmy stabilizace obrazu v reálném čase.

Video s názvem `shakyTrain.mp4` bylo následně stabilizováno dvěma různými metodami: jednak implementovaným online algoritmem, a dále referenčním offline algoritmem dostupným na platformě GitHub. Výstupy těchto stabilizačních procesů byly uloženy jako samostatné video soubory, přičemž stabilizace pomocí online algoritmu byla uložena pod názvem `online_stableTrain.mp4` a výstup z offline referenční metody byl uložen jako `offline_stableTrain.avi`. Testovací skript je spolu s veškerým zdrojovým kódem a se všemi vstupními videozáznamy uložen v přílohách této práce.

Tab. 4.1: Srovnání průměrného mezisnímkového posunu a směrodatné odchylky

Video	**Průměrný posun [px]**	**Směrodatná odchylka [px]**
shakyTrain.mp4	3.45	9.33
online_stableTrain.mp4	2.37	5.99
offline_stableTrain.avi	2.30	8.99

Je zřejmé, že navržené řešení je ve výchozím nastavení schopné účinně potlačit část nestability způsobenou pohybem kamery.

Z tabulky je patrné, že navržený algoritmus vykazuje vyvážený výkon – i když průměrná chyba je o něco vyšší než u offline metody. Naopak offline metoda vykazuje vyšší směrodatnou odchylku, což značí větší kolísání mezi jednotlivými snímky. To může být způsobeno tím, že offline metoda optimalizuje globální trajektorii, ale může přitom opomenout krátkodobé výkyvy.

Při vizuálním porovnání výsledků stabilizace je patrné, že video stabilizované pomocí online implementace působí díky agresivnější filtraci trajektorie konzistentněji, zejména ve druhé polovině testovacího záznamu.

Offline metoda však poskytuje stabilnější výsledek v úsecích s náhlými změnami pohybu, protože má předem k dispozici celou trajektorii pohybu. V takových situacích algoritmus fungující v reálném čase nemusí zareagovat okamžitě, což může vést k dočasnému nesouladu obrazu nebo jeho posunu mimo původní rámec.

4.2 Výkonnostní hodnocení

Tato část se zaměřuje na vyhodnocení výkonnosti implementovaného algoritmu digitální stabilizace videa. Cílem testování je ověřit, zda je navržený systém schopen zpracovávat vstupní snímky v reálném čase při zachování stabilní spotřeby výpočetních prostředků.

Měření výpočetní zátěže bylo provedeno na cílové výpočetní platformě Raspberry Pi 4B s kapacitou 8 GB RAM. Pro zajištění jednotných experimentálních podmínek byl jako zdroj obrazových dat použit videosoubor `shakyTrain.mp4` s rozlišením `640:360` pixelů a snímkovou frekvencí `25 fps`. Měření probíhalo s aktivovaným parametrem `measure_performance` a s ukládáním výsledků do příslušných výstupních souborů. V rámci měření byly sledovány následující metriky:

- **FPS** – průměrný počet snímků za sekundu,
- **zatížení CPU** – aktuální vytížení procesoru během běhu,
- **využití RAM** – spotřeba operační paměti algoritmem.

Experimenty byly provedeny za účelem posouzení výpočetní složitosti algoritmu v závislosti na konfiguraci parametrů. Byly identifikovány klíčové parametry, které mají nejvýraznější vliv na výpočetní zátěž:

- Parametr `resize_ratio`,
- Parametr `display_output`,
- Parametr `max_feature_count`,
- Rozlišení vstupního snímku.

Aby se minimalizoval vliv náhodné chyby, bylo každé měření při dané konfiguraci parametrů provedeno třikrát, přičemž uváděné výsledky představují průměrnou hodnotu ze tří opakování.

4.2.1 Vlivu parametru `resize_ratio`

Tab. 4.2: Závislost výkonnostních metrik na parametru `resize_ratio`

resize_ratio	FPS	CPU (%)	RAM (MB)	Drop rate (%)
0.25	120.3	216.2	249.5	100.0
0.50	63.7	190.2	251.5	0.19
0.75	42.6	176.1	252.2	0.06
1.00	32.5	159.9	253.1	0.13

Z výsledků uvedených v tabulce 4.2 vyplývá, že parametr `resize_ratio` má zásadní vliv na výpočetní náročnost algoritmu. Při zmenšení vstupních snímků na 25 % původní velikosti došlo k výraznému nárůstu hodnoty FPS, avšak současně k extrémnímu zvýšení hodnoty `drop_rate` na 100 %. Tento výsledek naznačuje, že systém sice dokáže velmi rychle zpracovávat menší snímky, avšak v této konfiguraci selhal detektor ORB a celá stabilizace se zcela rozpadla.

Při hodnotách `resize_ratio` v rozsahu `0.5-0.75` je dosaženo optimální rovnováhy mezi výkonností a nízkou ztrátovostí, což z této konfigurace činí vhodnou volbu pro provoz v reálném čase na platformě Raspberry Pi 4B.

Hodnota `resize_ratio = 1.0` znamená zpracování v plném rozlišení, což vede k nejnižšímu FPS. Tento kompromis může být vhodný v případech, kde je důležitá zachovaná vizuální kvalita a ztráta výkonu není kritická.

4.2.2 Vlivu parametru `max_feature_count`

Pro měření vlivu parametru `max_feature_count` byla použita následující konfigurace.

Tab. 4.3: Závislost výkonnostních metrik na parametru `max_feature_count`

max_feature_count	FPS	CPU (%)	RAM (MB)	Drop rate (%)
100	37.0	151.6	253.3	0.56
300	32.4	159.0	253.0	0.13
500	27.5	174.6	253.1	0.06

Jak ukazuje tabulka 4.3, parametr `max_feature_count` má přímý dopad na FPS a výpočetní zatížení. Při nejnižší hodnotě bylo dosaženo nejvyššího FPS a nejnižšího vytížení CPU, což odpovídá nižší výpočetní složitosti algoritmu při menším množství zpracovávaných prvků.

S navyšováním počtu klíčových bodů dochází ke snižování FPS, přičemž zároveň stoupá zatížení CPU. Tento výsledek je očekávaný, protože detekce, párování a zpracování většího počtu klíčových bodů představuje náročnější operace.

Z hlediska ztrátovosti je patrný opačný trend – při vyšším počtu bodů se ztrátovost snižuje. To ukazuje, že větší množství klíčových bodů zvyšuje robustnost detekce pohybu a stabilizace, a tedy i pravděpodobnost úspěšného zpracování snímku.

Celkově lze tedy říci, že parametr `max_feature_count` představuje důležitý nástroj pro vyvažování mezi výkonností a kvalitou stabilizace. Optimální hodnota závisí na konkrétních požadavcích aplikace.

4.2.3 Vlivu parametru `display_output`

Pro měření vlivu parametru `display_output` byla použita následující konfigurace.

Tab. 4.4: Závislost výkonnostních metrik na parametru `display_output`

display_output	FPS	CPU (%)	RAM (MB)	Drop rate (%)
`true`	25.1	144.3	280.1	0.13
`false`	32.3	161.2	253.0	0.13

Při aktivovaném zobrazování klesla FPS, což odpovídá očekávání — vykreslování obrazu na obrazovku je časově náročná operace, zejména na platformě s omezeným grafickým výkonem, jako je Raspberry Pi.

Zajímavým výsledkem je nižší průměrné vytížení CPU při zapnutém výstupu ve srovnání s režimem bez zobrazení. Tento jev lze vysvětlit tím, že CPU méně často zatěžováno hlavními výpočty algoritmu, protože více času tráví čekáním na dokončení operací spojených se zobrazením.

Při zapnutém zobrazení jsou kromě paměti potřebné pro zpracování snímků alokovány také prostředky pro jejich grafické vykreslení, a proto je celková paměťová náročnost vyšší.

Z těchto výsledků vyplývá, že pro dosažení maximálního výkonu v reálném čase je vhodné ponechat `display_output` vypnutý, zejména pokud je vizualizace nepotřebná pro běžný provoz systému.

4.2.4 Vlivu rozlišení vstupního snímku

Tab. 4.5: Závislost výkonnostních metrik na rozlišení vstupního snímku

Rozlišení	FPS	CPU (%)	RAM (MB)	Drop rate (%)
`320:180`	73.8	137.3	244.7	0.44
`640:360`	32.4	160.2	253.1	0.13
`960:540`	17.6	164.5	267.3	0.38
`1280:720`	10.9	164.4	285.9	0.31

Výsledky potvrzují očekávanou závislost mezi rozlišením vstupního videa a výkonností algoritmu. Se zvyšujícím se rozlišením dochází k výraznému poklesu FPS. Tento trend je důsledkem nárůstu počtu pixelů, které musí algoritmus zpracovat při každém snímku, což lineárně i kvadraticky zvyšuje výpočetní náročnost jednotlivých kroků – zejména detekce klíčových bodů, párování a výpočtu transformační matice.

základě provedených experimentů lze konstatovat, že výkonnost algoritmu digitální stabilizace videa je výrazně ovlivněna zvolenými parametry konfigurace. Za optimální konfiguraci pro provoz algoritmu v reálném čase na platformě Raspberry Pi 4B lze považovat použití rozlišení `640×360`, parametr `resize_ratio` v rozsahu 0,5–0,75, deaktivovanou vizualizaci výstupu a rozumný počet klíčových bodů v rozmezí 100–300. Tato kombinace parametrů zajišťuje vyvážený poměr mezi kvalitou stabilizace a výpočetní efektivitou.

Závěr

Cílem této práce bylo navrhnout, implementovat a experimentálně ověřit algoritmus digitální stabilizace videa, který je schopen běžet v reálném čase na platformě Raspberry Pi 4B. Tento cíl byl úspěšně splněn.

V praktické části byl vyvinut modulární systém napsaný v jazyce Python, který využívá detekci a párování klíčových bodů pomocí metody ORB a následné vyhlazování trajektorie kamerového pohybu pomocí jednoduchého Kalmanova filtru s konfigurovatelnými parametry. Součástí implementace je také mechanismus pro detekci statických scén, který zabraňuje akumulaci chyb při absenci pohybu a přispívá ke zvýšení celkové stability výstupu. Důraz byl kladen na reálnou použitelnost, přehlednost kódu a možnost snadné konfigurace jednotlivých parametrů.

Funkčnost algoritmu byla ověřena na testovacích videích a výsledky byly porovnány s původním nestabilizovaným záznamem a s referenční offline metodou. Kvantitativní vyhodnocení ukázalo, že navržený online algoritmus výrazně snižuje průměrný mezi snímkový posun i jeho variabilitu. Vizuálně působí stabilizované video plynuleji, zejména ve statických nebo mírně dynamických scénách. V extrémních situacích s náhlými pohyby kamery však dochází ke krátkodobé ztrátě korekce, což je typické omezení všech systémů pracujících v reálném čase.

Z dosažených výsledků vyplývá, že navržené řešení představuje vhodný základ pro nasazení reálné stabilizace na vestavěných zařízeních. V budoucnu lze algoritmus dále rozšířit například o stabilizaci měřítka, přechod k vícerozměrnému Kalmanovu filtru, který by lépe zachytil kovariance mezi parametry pohybu, nebo přejít ke homografickému modelu, který umožní kompenzovat složitější 3D transformace scény. Práce tak přináší nejen funkční a ověřenou implementaci, ale i otevřený prostor pro další technický rozvoj a experimentování.

Literatura

[1] Yiming Wang, Qian Huang, Chuanxu Jiang, Jiwen Liu, Mingzhou Shang, and Zhuang Miao. Video stabilization: A comprehensive survey. *Neurocomputing*, 516:205–230, 2023. [cit. 2024-12-15]. URL: https://www.sciencedirect.com/science/article/pii/S092523122201270X.

[2] Jan Novák, Anton Kaplanyan, Gabor Liktor, and Carsten Dachsbacher. Gpu computing: Image convolution. 2012. [cit. 2024-12-25]. URL: https://api.semanticscholar.org/CorpusID:61934636.

[3] NVIDIA. Fast corners detector overview. [online], 2024. [cit. 2024-12-25]. URL: https://docs.nvidia.com/vpi/algo_fast_corners_detector.html#algo_fast_corners_detector_overview.

[4] NVIDIA. Harris corners detector overview. [online], 2024. [cit. 2024-12-26]. URL: https://docs.nvidia.com/vpi/algo_harris_corners.html#algo_harris_corners_overview.

[5] Sung-Jea Ko, Sung-Hee Lee, and Kyung-Hoon Lee. Digital image stabilizing algorithms based on bit-plane matching. *IEEE Transactions on Consumer Electronics*, 44(3):617–622, 1998. [cit. 2024-12-27]. doi:10.1109/30.713172.

[6] Dianyuan Han. Comparison of commonly used image interpolation methods. In *Proceedings of the 2nd International Conference on Computer Science and Electronics Engineering (ICCSEE 2013)*, pages 1556–1559. Atlantis Press, 2013/03. [cit. 2024-12-31]. doi:10.2991/iccsee.2013.391.

[7] A. Balasundaram, S. Ashok Kumar, and S. Magesh Kumar. Optical flow based object movement tracking. *International Journal of Engineering and Advanced Technology (IJEAT)*, 9(1), October 2019. [cit. 2024-12-23]. URL: https://www.researchgate.net/publication/364095873_Optical_Flow_Based_Object_Movement_Tracking/fulltext/6360151b8d4484154a4f18ca/Optical-Flow-Based-Object-Movement-Tracking.pdf.

[8] Huan Yu and Wenhui Zhang. Moving camera video stabilization based on kalman filter and least squares fitting. In *Proceeding of the 11th World Congress on Intelligent Control and Automation*, pages 5956–5961, 2014. [cit. 2024-12-28]. doi:10.1109/WCICA.2014.7053740.

[9] Mohd. Ahmed. Digital video stabilization-review with a perspective of real time implemention. In *2017 International Conference on Recent Innovations*

in Signal processing and Embedded Systems (RISE), pages 296–303, 2017. [cit. 2024-12-15]. doi:10.1109/RISE.2017.8378170.

[10] Matthias Grundmann, Vivek Kwatra, and Irfan Essa. Auto-directed video stabilization with robust l1 optimal camera paths. In *CVPR 2011*, pages 225–232, 2011. [cit. 2024-12-15]. doi:10.1109/CVPR.2011.5995525.

[11] David Jacobs. Convolution and correlation. [online], 2023. [cit. 2024-1-05]. URL: https://www.cs.umd.edu/~djacobs/CMSC426/Convolution.pdf.

[12] OpenCV. Image pyramids. [online], 2024. [cit. 2024-12-25]. URL: https://docs.opencv.org/4.x/dc/dff/tutorial_py_pyramids.html.

[13] OpenCV. Fast algorithm for corner detection. [online], 2024. [cit. 2024-12-25]. URL: https://docs.opencv.org/3.4/df/d0c/tutorial_py_fast.html.

[14] OpenCV. Harris corner detection. [online], 2024. [cit. 2024-12-26]. URL: https://docs.opencv.org/4.x/dc/d0d/tutorial_py_features_harris.html.

[15] ScienceDirect. Feature descriptor. [online], 2024. [cit. 2024-12-26]. URL: https://www.sciencedirect.com/topics/computer-science/feature-descriptor.

[16] Michael Calonder, Vincent Lepetit, Christoph Strecha, and Pascal Fua. Brief: Binary robust independent elementary features. In Kostas Daniilidis, Petros Maragos, and Nikos Paragios, editors, *Computer Vision – ECCV 2010*, pages 778–792, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg. [it. 2024-12-26]. URL: https://link.springer.com/chapter/10.1007/978-3-642-15561-1_56.

[17] ScienceDirect. Hamming distance. [online], 2024. [it. 2024-12-26]. URL: https://www.sciencedirect.com/topics/computer-science/hamming-distance.

[18] NVIDIA. Orb feature detector. [online], 2024. [it. 2024-12-27]. URL: https://docs.nvidia.com/vpi/algo_orb_feature_detector.html.

[19] OpenCV. Feature matching. [online], 2024. [cit. 2024-12-27]. URL: https://docs.opencv.org/3.4/dc/dc3/tutorial_py_matcher.html.

[20] Konstantinos G. Derpanis. Ransac: Random sample consensus. [online], 2010. [cit. 2024-12-27]. URL: http://www.cs.yorku.ca/~kosta/CompVis_Notes/ransac.pdf.

[21] Eric Wolfram. Affine transformation. [online], 2024. [cit. 2024-12-28]. URL: https://mathworld.wolfram.com/AffineTransformation.html.

[22] Zhao Mm, Sun Jian, Sun Dihua, and Tang Yi. Fast tunnel monitoring video stabilization method based on improved orb feature. In *2018 Chinese Control And Decision Conference (CCDC)*, pages 5856–5861, 2018. [cit. 2024-12-28]. doi:10.1109/CCDC.2018.8408155.

[23] Feng Liu, Michael Gleicher, Jue Wang, Hailin Jin, and Aseem Agarwala. Subspace video stabilization. *ACM Trans. Graph.*, 30(1), February 2011. doi:10.1145/1899404.1899408.

[24] Marcos Roberto e Souza, Helena de Almeida Maia, and Hélio Pedrini. Survey on digital video stabilization: Concepts, methods, and challenges. *ACM Computing Surveys (CSUR)*, 55:1 – 37, 2022. URL: https://api.semanticscholar.org/CorpusID:246603335.

[25] Hung-Chang Chang, Shang-Hong Lai, and Kuang-Rong Lu. A robust real-time video stabilization algorithm. *Journal of Visual Communication and Image Representation*, 17(3):659–673, 2006. Special Issue on Real-Time Imaging. doi:10.1016/j.jvcir.2005.10.004.

[26] OpenCV Documentation. Optical Flow Tutorial. [online], 2024. [cit. 2024-12-26]. URL: https://docs.opencv.org/3.4/d4/dee/tutorial_optical_flow.html.

[27] Karina Mayen, Carlos Espinoza, Hugo Romero, Sergio Salazar, Mariano Lizarraga, and Rogelio Lozano. Real-time video stabilization algorithm based on efficient block matching for uavs. In *2015 Workshop on Research, Education and Development of Unmanned Aerial Systems (RED-UAS)*, pages 78–83, 2015. [cit. 2024-12-23]. doi:10.1109/RED-UAS.2015.7440993.

[28] M. Sushma Sri. Motion estimation and compensation techniques for video compression. *International Journal of Advanced Engineering Research and Science (IJAERS)*, 5(5):377–382, 2018. [cit: 2024-12-21]. URL: https://ijaers.com/uploads/issue_files/46-IJAERS-MAY-2018-43-Motion.pdf.

[29] Parth Bansal, Jahanvi B Dinesh, Shravan Kumar V. R., Sujay Krishna B, and T. S. Chandar. Video stabilization using orb detector. In *2022 14th International Conference on Computer and Automation Engineering (ICCAE)*, pages 50–55, 2022. doi:10.1109/ICCAE55086.2022.9762438.

[30] Greg Welch and Gary Bishop. An introduction to the kalman filter. Technical report, University of North Carolina at Chapel Hill, 2001. [cit. 2024-12-28]. URL: https://www.cs.unc.edu/~welch/media/pdf/kalman_intro.pdf.

[31] Eduard Sojka. *Digitální zpracování obrazu*. VŠB-Technická univerzita Ostrava, 2020. [cit. 2024-12-28]. URL: https://mrl.cs.vsb.cz/people/sojka/dzo/digitalni_zpracovani_obrazu.pdf.

[32] Jaroslava Frončková and Petr Pražák. *Matematické metody pro ekonomy*. Univerzita Hradec Králové, 2013. [cit. 2024-12-28]. URL: https://digilib.uhk.cz/bitstream/handle/20.500.12603/102/FRON%C4%8CKOV%C3%81%2C%20PRA%C5%BD%C3%81K.pdf?sequence=1&isAllowed=y.

[33] Andrey Litvin, Janusz Konrad, and William Clement Karl. Probabilistic video stabilization using Kalman filtering and mosaicing. In Bhaskaran Vasudev, T. Russell Hsing, Andrew G. Tescher, and Touradj Ebrahimi, editors, *Image and Video Communications and Processing 2003*, volume 5022, pages 663 – 674. International Society for Optics and Photonics, SPIE, 2003. [cit. 2025-1-1]. doi:10.1117/12.476436.

[34] Adam Spannbauer. python_video_stab. https://github.com/AdamSpannbauer/python_video_stab, 2019. Accessed: 2025-05-23.

Seznam příloh

A Obsah elektronické přílohy

Projekt je strukturován do několika modulů, které zajišťují jednotlivé části algoritmu digitální stabilizace videa. Následující výpis ukazuje organizaci souborů a složek v rámci přiloženého archivu:

```
/ ..... kořenový adresář projektu
├── stabilizer ..... modul pro stabilizaci videa
│   ├── __init__.py
│   ├── estimator.py
│   ├── frame_features.py
│   ├── smoother.py
│   └── transform.py
├── Videos ..... testovací a demonstrační videa
│   ├── offline_stableTrain.avi
│   ├── online_stableTrain.mp4
│   └── shakyTrain.mp4
├── config.json ..... konfigurační soubor
├── logger.py ..... modul pro logování běhu programu
├── run.py ..... hlavní spouštěcí skript
├── source.py ..... získávání snímků z videa nebo kamery
├── test.py ..... skript pro testování algoritmu
├── utils.py ..... pomocné funkce a nástroje
└── visualizer.py ..... vizualizace trajektorií a výstupů
```