ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE

Fakulta elektrotechnická

Katedra počítačů

Rozšíření a vylepšení KCF object trackeru

Extension and improvement of KCF object tracker

Bakalářská práce

Studijní program: Softwarové inženýrství a technologie

Vedoucí práce: Ing. Joel Matějka

Vypracoval: Jan Oravec DiS.

Ročník: 2019/2020

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Oravec** Jméno: **Jan** Osobní číslo: **457146**

Fakulta/ústav: **Fakulta elektrotechnická**

Zadávající katedra/ústav: **Katedra počítačů**

Studijní program: **Softwarové inženýrství a technologie**

II. ÚDAJE K BAKALÁŘSKÉ PRÁCI

Název bakalářské práce:

Rozšíření a vylepšení KCF object trackeru

Název bakalářské práce anglicky:

Extension and improvements of KCF object tracker

Pokyny pro vypracování:

1. Seznamte se se sledováním objektů v obrazu pomocí korelačních filtrů, konkrétní implementací a knihovnou OpenCV.
2. V dané implementaci navrhněte změny datových struktur a souvisejících částí kódu, aby byla zachována původní funkcionalita a zároveň využity datové struktury dostupné v OpenCV verze 4.2. Konkrétně využití UMat, Gmat a Graph API pro akceleraci výpočtů na GPU.
3. Implementujte navržené změny a průběžně testujte jejich správnost. Porovnejte výkonnost jednotlivých implementací bežících na CPU a GPU.
4. Řádně zdokumentujte provedené změny.

Seznam doporučené literatury:

[1] João F. Henriques, Rui Caseiro, Pedro Martins, Jorge Batista, “High-Speed Tracking with Kernelized Correlation Filters“, IEEE Transactions on Pattern Analysis and Machine Intelligence, 2015
[2] Kernelized Correlation Filter tracker. GitHub [online]. Dostupné z: https://github.com/CTU-IIG/kcf
[3] OpenCV documentation. OpenCV [online]. [cit. 2020-01-08]. Dostupné z: https://docs.opencv.org/4.2.0

Jméno a pracoviště vedoucí(ho) bakalářské práce:

Ing. Joel Matějka, katedra řídicí techniky FEL

Jméno a pracoviště druhé(ho) vedoucí(ho) nebo konzultanta(ky) bakalářské práce:

Datum zadání bakalářské práce: **12.02.2020** Termín odevzdání bakalářské práce: **22.05.2020**

Platnost zadání bakalářské práce: **30.09.2021**

Ing. Joel Matějka podpis vedoucí(ho) práce	podpis vedoucí(ho) ústavu/katedry	prof. Mgr. Petr Páta, Ph.D. podpis děkana(ky)

III. PŘEVZETÍ ZADÁNÍ

Student bere na vědomí, že je povinen vypracovat bakalářskou práci samostatně, bez cizí pomoci, s výjimkou poskytnutých konzultací. Seznam použité literatury, jiných pramenů a jmen konzultantů je třeba uvést v bakalářské práci.

. Datum převzetí zadání

Podpis studenta

Prohlášení:

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací.

V Praze, dne 17.5.2020 Podpis autora: ……..

Poděkování:

Děkuji vedoucímu bakalářské práce Ing. Joel Matějkovi, za vstřícnost a cenné rady při zpracování této práce.

Abstrakt

Tato práce se zabývá problematikou rozšíření a optimalizace konkrétní implementace object trackeru založeného na korelačních filtrech. Pod pojmem rozšíření a optimalizace je myšlena aktualizace zastaralých částí programu, především s ohledem na datové struktury dostupné v novějších verzích využívaných knihoven.

Zvolená implementace je postavená na hojně používané knihovně OpenCV, která se specializuje na výpočetní úkony s maticemi, reprezentujícími obrazová data. Bakalářská práce zahrnuje kromě vlastních změn v kódu do značné míry studium dostupných metod knihovny OpenCV, nezbytných pro správné využití, hloubkovou analýzu kódu dané implementace, a sestavení plánu konverze. Samozřejmostí je zachování stávající funkcionality kódu i po aplikaci změn, proto je nezbytné průběžné testování shody výstupních hodnot původní a nové verze aplikace.

Tento přístup přinesl úspěch při postupné konverzi kódu pro použití s efektivněji pracujícími datovými typy a rozhraními knihovny OpenCV. Výsledkem bakalářské práce je upravený program produkující stejné výsledky jako implementace původní, a neobsahující žádný zastaralý kód. Program nyní také dokáže pracovat podstatně rychleji, ale míra tohoto zrychlení byla do určité míry snížena omezeními platnými pro použité datové typy a rozhraní knihovny OpenCV.

Vzhledem k současnému vývoji knihovny OpenCV lze předpokládat, že daná omezení budou odstraněna a bude možné současnou implementaci ještě vylepšit.

Slovník pojmů

Tracking – Činnost udržování přehledu o poloze obdélníkové výseče videa při jeho běhu.

KCF Tracker – Program používaný k trackingu polohy objektu v běžícím videu.

Počítačové vidění – Soubor metod, na jejichž základě je počítač schopen rozeznat přítomnost předmětů v obrazových datech, a dále s touto informací pracovat.

OpenCV – Open source knihovna, specializující se na implementaci metod počítačového vidění.

VOT challenge – Projekt, jehož cílem je vývoj aplikací, pracujících s konceptem počítačového vidění

Abstract

This document explores the problematic of expanding and improving a specific implementation of object tracker, which is based on correlation filters. Tracker uses methods of computer vision to track position of a defined object in a continuing sequence of picture frames. The term of expansion and improvement is meant to describe actualization of deprecated parts of the program, especially in regard to data structures available in newer versions of used libraries.

The chosen implementation is built on a popular library OpenCV, which specializes on computing operations that involve matrices of graphical data. Aside from editing code of the program, this bachelor project involves to great extent studying of available methods in library OpenCV, which is necessary for proper use, analysis a design of the planned conversion. It is of course necessary to ensure unchanged functionality of the original implementation, which is why gradual testing and comparison of output values is required between the new and previous versions of the program.

This approach brought success while converting code of the program for use with more effectively working data types and interfaces of library OpenCV. The result of this bachelor project is edited program that produces same results as the original implementation, and that is free of deprecated code. Program is also able to work substantialy faster than before, but the extent of this improvement was somewhat decreased by limitations of used data types and interfaces from library OpenCV.

In future, optimalization of the program through the use of this library will likely become possible, as OpenCV is still in development and its current limitations are likely to be resolved.

List of definitions

Tracking – Process during which location of a chosen rectangle area in a video is being kept track of.

KCF Tracker – Program used for tracking, and the subject of this bachelor project.

Computer vision – Set of methods, which enable a computer to recognise presence of physical objects in graphical data, and then further process that information.

OpenCV – Open source library, specializing in implementation of computer vision methods.

VOT challenge – Project whose purpose is development of aplications, that use the concept of computer vision.

Obsah

1) Úvod

Účelem této bakalářské práce je optimalizace a rozšíření programu KCF Tracker, za účelem odstranění zastaralých částí kódu a zvýšení rychlosti zpracování vstupních dat. Toho by mělo být dosaženo především nahrazením tříd ComplexMat a DynMem třídou cv::Mat, která je součástí knihovny OpenCV. Dalšími úkoly jsou konverze typu matic používaných v programu na cv::UMat a implementace maticových výpočtů pomocí rozhraní Graph API, které jsou opět součástí OpenCV.

V tomto dokumentu se čtenář může seznámit s teorií a strukturou implementace KCF tracker, postupem jeho optimalizace, a výsledky měření průměrných rychlostí zpracování vstupních dat pro každou dokončenou část optimalizace.

V první části zprávy je popsán princip fungování programu KCF tracker. Je zde objasněn koncept korelačních filtrů a způsob jejich použití v rámci projektu. Následuje obecný úvod do knihovny OpenCV, jejího účelu a nejvíce používaných součástí. Na konci této části je popsána organizační struktura kódu trackeru a workflow jeho procesů.

Po představení termínů dané problematiky následuje návrh plánu zpracování bakalářské práce. Je zde obsažena úvodní analýza implementace programu, doporučená obecná metodika a návrhy zpracování jednotlivých úkolů bakalářské práce.

Další kapitola je zprávou popisující změny, které byly provedeny, jejich průběh a konečné výsledky. Jsou zde také výsledky měření rychlosti programu v každé fázi zpracování bakalářské práce, a vyvozené závěry.

Závěrem jsou popsány dosažené výsledky v rámci celého projektu, významné nálezy a seznam referencí použitých v průběhu sepsání této zprávy.

Poslední částí zprávy jsou přílohy, obsahující tabulky naměřených rychlostí programu, v zájmu zachování přehlednosti této zprávy.

2) Teoretický základ

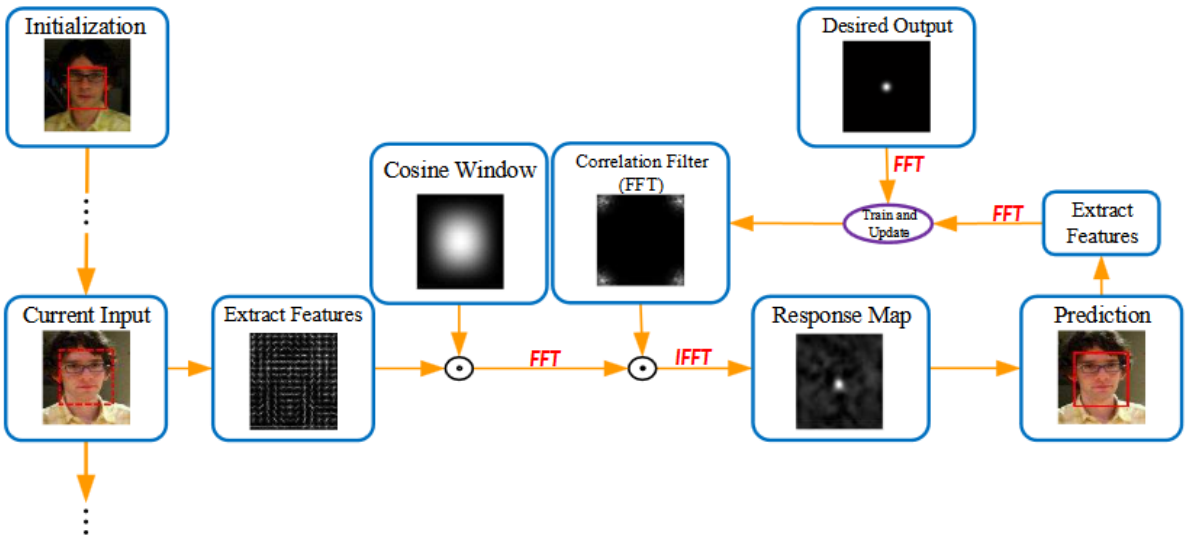
V této sekci bude postupně představena aplikace KCF trackeru, její komponenty a využité knihovny v rozsahu nezbytném k pochopení záběru bakalářské práce.

Sledování objektů s využitím korelačních filtrů

Object tracker je počítačový program, který má za úkol udržovat přehled o poloze vybrané části obrazu ve video streamu. Činnost tohoto sledování se nazývá „tracking“.

Jednou z možností, jak definovat oblast zájmu ke sledování, je výběr obdélníku v grafickém prostředí pomocí myši. Program v průběhu videa odhaduje novou polohu oblasti, kterou pak uživatel může sledovat s pohybem předtím vybraného obdélníka.

Vysvětlení principu fungování korelačních filtrů v této práci je citováno z disertační práce [1] vytvořené Tomášem Vojířem, který je autorem programu KCF Tracker.



Obr č.1 – Workflow trackeru založeném na principu korelačních filtrů [1]

Tracker je inicializován počátečním snímkem v sérii, který je použit k vytvoření počátečního korelačního filtru. Ten je založen a upravován takovým způsobem, aby jeho výstup dosáhl co nejlepší shody s tzv. oknem Gaussova vrcholu po aplikaci na označenou oblast ve snímku.

Pro každý následující snímek jsou pak označeny výrazné prvky ve sledované oblasti, a je aplikováno kosinové okno pro zjemnění kontrastu prvků na hranicích zabírané oblasti. Výsledek této operace je převeden do Fourierovy domény a je na něj aplikován filtr získaný z předchozího snímku. Nakonec je aplikována inverzní Fourierova transformace pro získání mapy nejlepší shody, a nejlepší výsledek v mapě je pak použit k upravení pozice sledované oblasti. Ze současného snímku je vytvořen nový korelační filtr, a začíná nová smyčka trackingu.

Protože jednotlivý filtr nemusí vždy vrátit dostatečně dobrou shodu, je v této implementaci object trackeru vytvářeno pro každou zpracovávanou oblast celkem 15 korelačních filtrů, testujících na různé náklony a oddálení sledovaného objektu.

Knihovna OpenCV

Jde o dobře zavedenou a open source knihovnu, obsahující metody a třídy pro implementaci počítačového vidění [2]. Vzhledem k její spolehlivosti a všestrannosti byla zvolena pro práci maticovými daty této bakalářské práce.

Nejpoužívanější třídou knihovny je cv::Mat, která reprezentuje maticová data. Tato třída podporuje více kanálovou reprezentaci dat, což se používá například při ukládání RGB hodnot pro každý pixel obrazu. Je také možné definovat více dimenzionální matice pro práci s několika maticemi nižšího řádu v jedné instanci cv::Mat.

Třída cv::Mat však není samotným nosičem dat, ale pouze hlavičkou která na ně odkazuje a popisuje jejich strukturu. Copy constructor tedy produkuje identickou hlavičku, která odkazuje na stejná data jako originál, a provádí na stejných datech operace.

Pokud je potřeba pracovat nad samostatnou kopií dat bez poškození originálu, je možné použít metodu cv::Mat.clone(). Tento způsob je ale výkonnostně náročnější, a neměl by být používán opakovaně v sérii, aby nedošlo k poklesu rychlosti běhu programu.

Vzhledem k povaze zadání této práce stojí za zmínku funkce cv::Mat.ptr<_Tp>(). To je funkce umožňující přímý přístup k datům v dimenzi matice, dané jako vstupní argument funkce. Typ <_Tp> potom udává, jak chceme interpretovat data v matici. Je to argument který umožňuje omezenou typovou konverzi, a našem případě především interpretaci dvou sousedících hodnot v matici jako typ std::complex . To je neocenitelná vlastnost především při konverzi matic obsahující komplexní čísla, které jsou v projektu.

Důležitou součástí metod obsažených v knihovně OpenCV je také funkce cv::dft(), která umožnuje převod matic pomocí Fourierovy transformace, což výrazně snižuje čas potřebný ke zpracování velkého množství obrazových dat. To je vlastnost hojně využívaná při práci s obrazovými daty v programu.

Tato metoda je schopná dopředné i zpětné transformace na základě konstanty vložené jako vstupní argument funkce.

Poslední funkcí s velkou použitelností v následujících úpravách je cv::mixChannels(). Jak název napovídá, jde o metodu umožňující přesunout hodnoty v kanálu jedné matice do kanálu druhé. Je tedy možné například postupně vkládat jednokanálové matice do jiné matice sloužící jako úložiště, a pak s touto jednou maticí pracovat jako se souborem matic.

Rozhraní Graph API

Toto rozhraní je poměrně novým přírůstkem knihovny OpenCV umožňující explicitní odesílání výpočetních úkonů do procesoru GPU. Konečným cílem této práce je použití implementace tohoto rozhraní v programu za účelem zvýšení jeho efektivity.

Úkolem tohoto rozhraní je vytvářet balíčky operací nad vstupními maticovými daty, vnitřně reprezentované třídou cv::GMat, které je možné opakovaně posílat na GPU s novými vstupními parametry [3]. Dalo by se říct, že výsledné balíčky jsou po částech sestavenými lambda funkcemi, se specializací pro práci s matičními výpočty.

Použití rozhraní Graph API tak umožňuje přesun některých výkonnostně náročných operací z procesoru počítače na GPU, čímž se uvolní část jeho kapacity.

Struktura upravované aplikace

Program KCF Tracker [4] je napsaný v jazyce C++, a kompilovaný pomocí nástroje CMake. Projekt obsahuje několik sestavitelných verzí, lišícími se použitou knihovnou pro výpočet Fourierovy transformace, a požadovanou funkcionalitou. Tato práce se zabývá verzí VOT, kde inputem programu je série snímků videa, s podporou výpočetních knihoven Fftw3 a OpenCV.

Třída, která se stará o hlavní funkci aplikace, se nazývá KCF_Tracker. Zde jsou umístěny všechny hlavní funkce a konfigurační proměnné ovlivňující chování programu.

Jeho součástí je třída Model, která uchovává informace o právě sledovaném snímku.

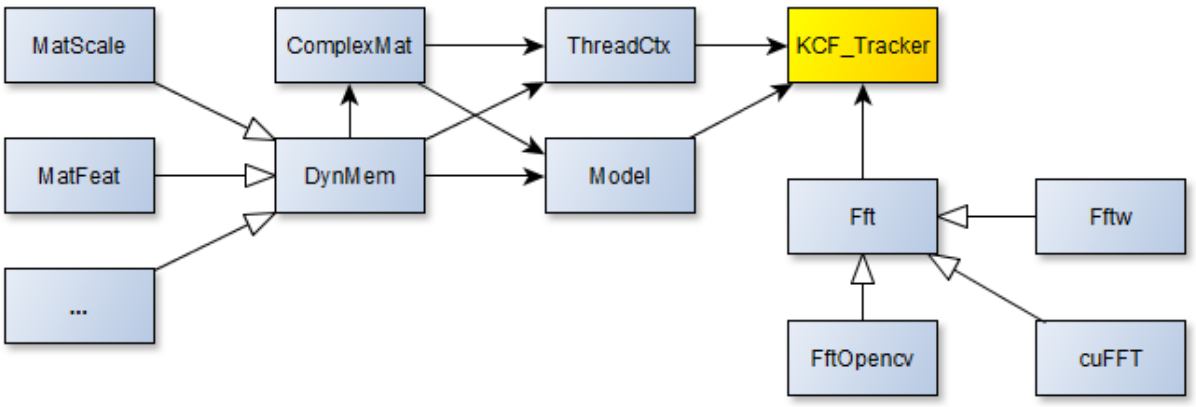
Program pracuje na základě porovnávání sledované oblasti s novým snímkem. Při každém porovnání je vytvořeno 15 různě vychýlených verzí sledované oblasti, které jsou více či méně nakloněné a oddálené od originálu. Tyto verze jsou reprezentované třídou ThreadCtx obsahující všechny výsledky výpočtů pro určení nejlepší shody s dalším snímkem.

V době implementace KCF trackeru nebyly v OpenCV dostupné třídy pro reprezentaci komplexních čísel, a proto bylo definováno několik dalších podpůrných tříd popsaných níže.

ComplexMat je třída kombinující implementaci cv::Mat a dynamického pole komplexních čísel. Je používána jako proměnná v ostatních třídách vždy, když je třeba provádět operace na matici komplexních hodnot.

DynMem je jednoduchá třída chovající se jako dynamické pole s nastavitelným typem. Je použita jako úložná proměnná třídy ComplexMat, a na jejím základě existuje několik rozšiřujících podpůrných tříd určených pro ukládání specifických formátů dat, například MatScales nebo MatFeats.

Třída Fft obsahuje malé množství proměnných potřebných k výpočtu Fourierovy transformace, jejím hlavním účelem je ale kontrola. Všechny metody této třídy se skládají z assertů, pomocí kterých se kontroluje formát vstupních matic pro metody skutečně provádějící Fourierovu transformaci. Tyto metody jsou obsažené v třídách implementující Fourierovu transformaci pomocí různých knihoven.



Obr. 1 – Hiearchie tříd programu

Workflow aplikace

Vstupním bodem programu je soubor „main_vot.cpp“. Po spuštění funkce main() jsou přečteny parametry zadané do příkazové řádky při spuštění programu, a podle nich nastaveny vnitřní proměnné třídy KCF_tracker.

Jednou z důležitějších funkcí následujících po tomto kroku je KCF_Tracker::init(), která připraví program k porovnání prvního snímku. To zahrnuje hlavně inicializaci třídy Model, instancí ThreadCtx, a podpůrné třídy GaussianCorrelation, která se chová jako funkce závislá na vnitřním stavu.

Inicializace KCF_Tracker potom přímo navazuje na funkci KCF_Tracker::train(), která zaplní třídu Model informacemi potřebnými k porovnávání s prvním snímkem videa.

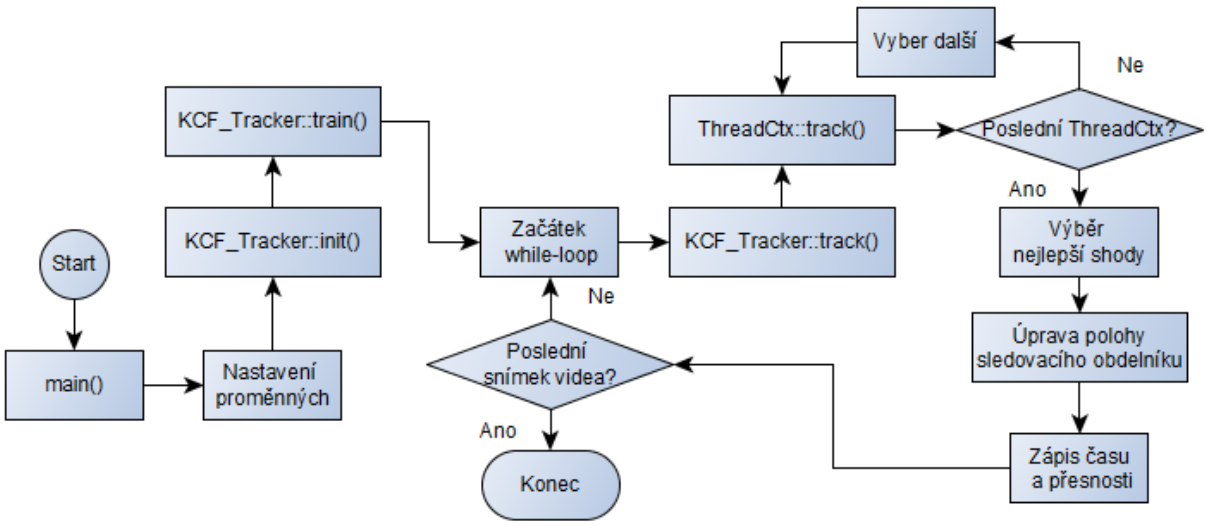
Po dokončení inicializace se program vrací na úroveň metody main(), a hned po té spouští sledovací smyčku while-loop. Tato smyčka se stará o vlastní tracking sledované oblasti, a běží po celou dobu života programu.

Tracking se odehrává opakovaným voláním funkce KCF_Tracker::track(). Ta si z interního úložiště KCF_Tracker vezme uložené instance ThreadCtx předchozího snímku a zavolá na nich metodu ThreadCtx::track(). Ta vypočítá, která z jejich kombinací náklonu a zoomu má nejlepší shodu se současným snímkem na základě informací v třídách ThreadCtx a Model, a uloží ji jako číselnou hodnotu. V průběhu výpočtu jsou také nastaveny proměnné instance ThreadCtx aby odpovídaly současnému snímku.

ThreadCtx s nejlepší shodou je pak vybrán, a na jeho základě pak určena nová poloha sledované oblasti snímku.

Na konci metody KCF_Tracker::track() je pak opět zavolána metoda KCF_Tracker::train(), která nastaví informace v třídě Model aby odpovídaly současnému snímku, a daly se použít k porovnání se snímkem co bude následovat.

Program se poté vrací zpět na úroveň metody main(), zaznamená přesnost a rychlost výpočtu do logu, a začne další kolo smyčky while.



Obr. 2 – Průběh činnosti programu

3) Návrh řešení

Úvodní analýza problému

Prvním úkolem v rámci této práce bylo seznámení se strukturou upravovaného programu a využívanou knihovnou OpenCV. Samotná implementace KCF trackeru je poměrně rozsáhlá a komplikovaná, proto plánované úpravy kódu vyžadují důkladnou analýzu.

Po uchopení podstaty a struktury programu je třeba navrhnout nejlepší přístup při provádění konverze z tříd ComplexMat a DynMem na cv::Mat. Struktura těchto tříd je podobná, ale ne úplně shodná s cv::Mat, takže je potřeba navrhnout nový způsob reprezentace původních dat v novém formátu na srovnatelné úrovni.

Samotné úpravy výskytu původních tříd jsou navíc také složité, protože na nich závisí velká část kódu, který maticové proměnné používá různými způsoby. To znemožní běh programu, dokud nejsou implementovány manipulační metody počítající s novou strukturou. Je proto nezbytné navrhnout způsob práce, který tento problém nevyvolá.

Obecná metodika řešení

Navrhovaným postupem implementace konverzí je vytvoření duplicitních maticových proměnných ve všech důležitých třídách, nad kterými budou vykonávány a testovány nově definované operace. Tyto nové operace budou také volány duplicitně vedle původních funkcí, čímž bude docházet k paralelnímu zpracování dvou souborů dat: Testovacích a Aktivních.

Tyto duplicitní testovací proměnné budou inicializovány stejným způsobem jako aktivní proměnné, a konverze operací nad nimi bude vytvářena postupně ve směru zpracování kódu programu.

Tímto způsobem je možné testovat správnost nové implementace ihned po dokončení konverze, a nedojde k přerušení funkčnosti kódu. Původní funkce navíc zůstanou zachovány v původním tvaru pro účely porovnání a analýzy jejich původního účelu.

Duplicitní proměnné dostanou stejné jméno jako původní proměnné, ale na konec jejich jmen bude přidán postfix „_Test“.

Po dokončení konverze jednoho typu matice v celém kódu je pak možné tento postfix odstranit, a proměnné i volání metod původního typu smazat. Tím prakticky dojde k přenesení funkční závislosti ze staré implementace na novou.

Tato obecná metodika bude použita u všech stupňů typové konverze v projektu.

Konverze na cv::Mat

Po dokončení těchto příprav je možné začít s prvním skutečným zpracováním kódu. Třídy ComplexMat, DynMem a jejich dědičné třídy musí být odstraněny z kódu předtím, než bude možné provádět další změny. Ideálně by mělo být dosaženo stavu, kdy je možné odstranit

zdrojový kód těchto tříd bez chyby programu, a bez negativního dopadu na kvalitu výstupu programu.

Po dokončení tohoto kroku a kontrole správnosti výstupů bude porovnána rychlost programu před a po implementaci konverze.

Konverze na cv::UMat

Místa s velkou výpočetní a časovou náročností pak budou dále konvertována z typu cv::Mat na cv::UMat. Jedná se o mezikrok zjednodušující další konverzi kódu pro použití s cv::GMat, následující později. Třída cv::UMat je funkčně stejná jako cv::Mat, ale navíc dokáže automaticky vybrat mezi typem procesorové jednotky zpracovávající zadané úkoly. Pokud je to tedy optimálnější, může implicitně svěřit výpočetní úkon procesoru GPU místo CPU.

Důvodem tohoto mezikroku je právě povrchní otestování chyb, které mohou vyskytnout při přenášení výpočetních úloh z CPU do GPU.

Konverze na implementaci GraphAPI

Pro dokončení konverze na Graph API je potřeba najít všechna místa v kódu kde dochází k náročnějším výpočetním operacím, a nahradit je implementací tohoto rozhraní. Množina těchto míst je však mnohem užší než u předchozích dvou konverzí, neboť Graph API zatím poskytuje pouze omezený rozsah operací, které lze vložit do nově vytvořených balíčků.

Všechny výstupy instrukcí definovaných v Graph API jsou navíc pouze celé matice nebo seznam celých matic, takže nelze provádět jemnější operace nad menšími částmi vstupních dat, a jejich výsledek potom použít u operací co následují.

Navrhovaným řešením je tedy postupovat kódem aplikace ve směru jeho zpracování, a nahradit všechna místa které splňují podmínky proveditelnosti a výpočetní náročnosti.

Konverze Graph API pro Fourierovy transformace

Posledním implementačním úkolem je konverze Fourierových transformací na jejich implementaci v Graph API. Důvodem proč tato konverze nebyla pokryta v předchozí podkapitole je ten, že v době zpracování této bakalářské práce neexistuje jejich definice v rozhraní Graph API. Existují však nástroje, pomocí kterých lze vytvořit vlastní funkce v jeho kontextu.

Knihovna OpenCV obsahuje rozhraní Kernel API [5], který umožňuje deklarovat funkce v kontextu Graph API, a následně k nim přiřadit definici vhodnou pro daný problém.

Tato implementace navíc může být napsána v libovolném programovacím jazyce, pro který existuje definovaný Kernel API backend, zodpovědný za překlad kódu z jiného jazyka. Lze tak použít vhodnější řešení z jiných programovacích prostředí navzdory jejich odlišnosti, a dosáhnout ještě větší míry optimalizace.

Kromě C++, což je jazyk použitý pro implementaci knihovny OpenCV, je možné definovat řešení pomocí jazyků OpenCL a Halide. Tyto jazyky jsou často používány právě pro implementaci řešení zahrnující práci s grafickými daty.

Pokud je výhodnější použít jiné programovací jazyky, lze definovat nové backendy opět pomocí rozhraní Kernel API.

Navrhovaným řešením pro tuto část bakalářské práce je deklarace nových funkcí v kontextu Graph API, a jejich definice pomocí knihoven OpenCV a fftw3. Tyto knihovny byly původně použity v programu pro implementaci Fourierovy transformace, a obě používají jazyk C++, takže není třeba definovat Kernel API backend.

Průběžná kontrola

Všechny výše zmíněné implementační cíle bakalářské práce je třeba průběžně testovat. Zkoušenými kritérii jsou především shodnost hodnot nových maticových dat s originálem, bezproblémovost kompilace kódu, integrita maticových dat po použití nových metod, a absence warningů.

4) Implementace a výsledky

Úvodní příprava pracoviště

V rámci bakalářské práce byl obdržen přístup ke Git repozitáři obsahující originální kód programu. Z tohoto zdroje byl vytvořen nový osobní repozitář [6], který byl dále používán jako úložiště všech dalších změn v projektu. Výsledky těchto úprav pak byly hromadně posílány do repozitáře originálního kódu pomocí funkcionality „Pull request“, po schválení vedoucím práce.

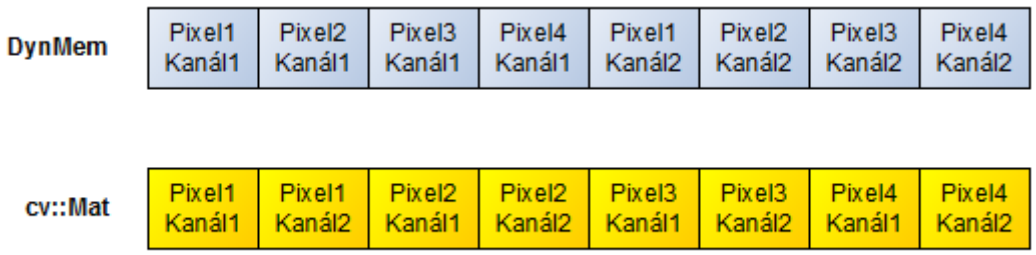
Po několika dotazech na vedoucího, a zkouškách na osobním laptopu, bylo zjištěno, že kód programu lze spustit pouze na operačním systému Linux. Protože osobní laptop původně určený k manipulaci s kódem používá systém Windows, bylo nakonec dohodnuto, že k vypracování bakalářské práce bude lepší použit zapůjčený laptop.

S vedoucím práce pak byla dále vyřešena otázka nastavení pracovního prostředí a vývojářského nástroje pro práci s knihovnou OpenCV.

Analýza kódu

Na začátku práce byl prostudován kód programu a učiněn pokus o návrh plánu pro zpracování potřebných úprav. Po provedení těchto kroků byly provedeny první zkušební konverze metod třídy ComplexMat, ale ty se nakonec ukázaly jako neefektivní.

Konverze metod přímo pro použití s novou třídou nemohlo být efektivní, protože cv::Mat řadí uložená data ve vnitřním poli jiným způsobem, než původní datové struktury. Původní algoritmy zacházení s daty pracující s tímto předpokladem proto nemohly být použity tak jak byly, a musely být přepracovány.



Obr3. – Porovnání formátu ukládání maticových dat

Proto byla po tomto počátečním pokusu provedena hloubková analýza formátu dat v třídách ComplexMat, DynMem, a jejich odvozených typech, s cílem najít jejich vhodné ekvivalenty. Stejný druh analýzy se týkal také manipulací s nimi a jejich zamýšlených účelů.

Jedním ze způsobů této analýzy se stalo vytvoření „implementačního hřiště“, umožňujícího sledování účinků manipulačních metod na instance třídy cv::Mat a ComplexMat. Program byl schválně předčasně ukončen zavoláním příkazu „return 0;“, a v prostoru před tímto voláním pak probíhaly všechny analytické testy.

Tento prostor vyžadoval minimum potřebné práce pro vytvoření, a po dokončení práce bylo možné ho snadno zrušit uzavřením testovacího kódu do komentářových závorek, a odstraněním příkazu „return 0;“.

Po vytvoření tohoto programátorského nástroje se stala analýza mnohem snazší než předtím, protože bylo možné vypsat obsah instancí v kontrolovaném prostředí a lépe pozorovatelném měřítku. Stejně tak odpadla starost s hledáním výpisů v konzoli, protože k jejich volání došlo na začátku programu, místo uprostřed nebo v průběhu smyčky while.

Navzdory těmto vylepšením je však celý projekt poměrně rozsáhlý a komplikovaný, a proto nakonec celá analýza zabrala víc jak čtvrtinu času potřebného ke zpracovaní současného stavu bakalářské práce.

Důsledkem této analýzy ale nakonec vznikla ucelená představa o potřebných změnách, stejně jako obecná metodika práce, popsaná v předchozí kapitole (viz. Návrh řešení). Tato metodika byla následně použita při konverzi z původních typů ComplexMat a DynMem na typ cv::Mat.

Mimo jiné byl touto analýzou získán přesnější přehled o průběhu zpracování kódu, popsaném v podkapitole Workflow aplikace, kapitoly Teoretický základ (viz.výše).

Tento přehled obsahuje přehled funkcí, nejvíce se podílejících na běhu programu. Tyto funkce jsou následující:

- Funkce KCF_Tracker::init()
    - Zodpovědná za počáteční nastavení vnitřního stavu programu
- Funkce KCF_Tracker::train()
    - Zodpovědná za uložení výpočetních informací o současném snímku, podle kterých se později počítá nejlepší shoda s následujícím snímkem
- Funkce KCF_Tracker::track()
    - Pro výpočet nejlepší shody mezi předchozím a následujícím snímkem
- Funkce ThreadCtx::track()
    - Pro vyjádření míry shody pomocí číselné hodnoty, na základě dat připravených ve funkci KCF_Tracker::track()
- Funkce KCF_Tracker::GaussianCorrelation::operator()
    - Používaná při určování shody snímků
    - Funkce závislá na vnitřním stavu maticových proměnných
- Všechny funkce pro manipulaci maticových dat ve třídě ComplexMat
    - V rámci odstranění třídy ComplexMat byly jejich konverze přesunuty do souboru „matutil.h“
- Všechny implementace Fourierových transformací
    - Funkce definovány v souborech fft_<knihovna>.cpp

Další důležitou informací je také seznam tříd důležitých pro běh programu. Tyto třídy obsahují vnitřní maticové proměnné zodpovědné za správný výstup programu, a jsou zároveň hlavním cílem konverzí bakalářské práce, hned vedle manipulačních funkcí.

Těmito třídami jsou především následující:

- KCF_Tracker
- KCF_Tracker::Model
- KCF_Tracker::GaussianCorrelation
- ThreadCtx

Konverze na cv::Mat

Tato část konverze vycházela z analýzy kódu a obecné metodiky popsaných v předchozí podkapitole (viz. Analýza kódu). Výchozí kód byl po celou dobu úprav ponechán v původním stavu a funkčnosti, a všechny proměnné tříd důležitých pro běh programu byly duplicitně deklarované v novém typu, a s postfixem „_Test“. Tyto testovací proměnné prošly na začátku zpracování kódu stejnou inicializací jako ty původní, a volání nově konvertovaných funkcí bylo prováděno paralelně s původní implementací programu.

Třída ComplexMat původně obsahovala všechny metody manipulace se svým formátem maticových dat, které se nevztahovaly k žádné jiné třídě v programu. V zájmu lepší čitelnosti kódu tak byly nově definované konverze těchto metod vloženy do nově vytvořeného souboru „MatUtil.h“ jako statické funkce nevyžadující existující instance nadřazené třídy k jejich zavolání.

Všechny probíhající konverze metod a typů byly postupně zpracovány, počínající v místě první inicializace proměnných, a postupující dále ve směru vykonávání aplikace (viz. kapitola Workflow aplikace). Každá dílčí konverze potom prošla porovnáním s původní implementací v zájmu zachování integrity dat a správnosti výstupu programu.

Tento způsob testování například pomohl odhalit chybu při použití operátorových metod, kde byl omylem ponechán zápis do původní matice místo vytvoření nové kopie.

Po dokončení všech úprav byla přepnuta závislost výstupu programu z původních proměnných na testovací, a otestována správnost sledovací funkce programu, stejně jako výstupního stavu vnitřních proměnných.

Oba tyto testy prošly bez problémů.

Dále prošla nová implementace porovnáním průměrné rychlosti zpracování jednotlivých snímků.

Ze stránek projektu VOT [7] byly staženy zkušební soubory snímků, na kterých je možno testovat schopnosti programu provádět tracking. Konkrétně šlo o snímky tří videozáznamů s titulky „bag“ (plastový sáček unášený větrem), „ball1“ (fotbalový míč se kterým si hraje student před budovou školy) a „basketball“ (sportovec během basketbalového utkání).

Vstupní podmínky a obecný formát testování byl zvolen následovně:

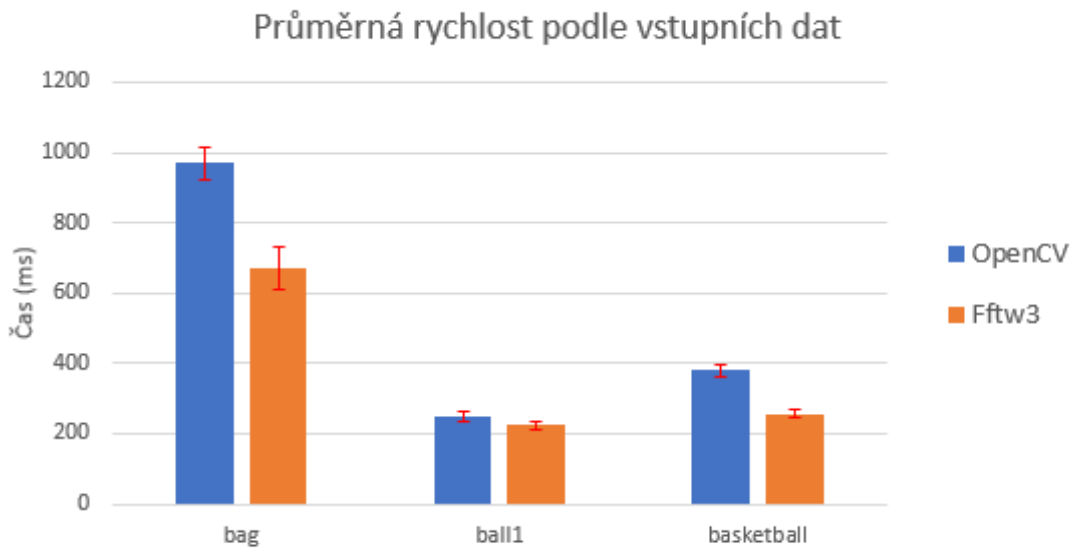
- Každý dokončený stupeň konverze projektu bude postupně otestován každým ze tří souborů snímků, pro otestování rychlosti programu při použití různých druhů testovacích dat.
- Test každého souboru snímků proběhne 20 krát pro každou typovou konverzi projektu, zvlášť pro každou implementaci podle knihoven OpenCV a Fftw3.
- Každá testovaná instance spuštění programu bude sledovat stejnou počáteční pozici a velikost trackované oblasti, zadanou pevně jako vstupní argument programu (viz. tabulka níže).
- Program bude spouštěn bez vstupního argumentu –visualize, který umožňuje zobrazení průběhu trackingu na obrazovce.
- Všechny testy budou probíhat na zařízení s následujícími technickými parametry:
    - OS: Ubuntu 16.04 LTS 64-bit
    - RAM paměť: 15.4 GB
    - Procesor: Intel Core i7-8550U, 8 x 1.8 GHz
    - Grafická karta: Intel UHD Graphics 620

Tabulka použitých vstupních údajů, udávajících počáteční pozici a velikost sledované oblasti je následující:

Soubor snímků	X	Y	šířka	výška
bag	345	180	80	90
ball1	490	415	50	55
basketball	190	225	35	80

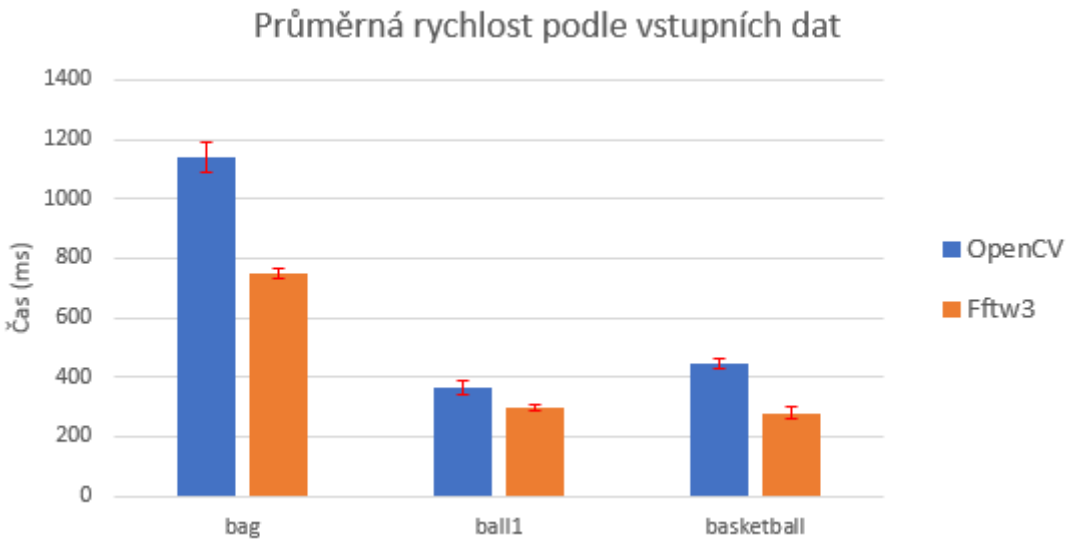
S použitím výše popsaného formátu byla otestována původní implementace programu a konvertovaná verze používající cv::Mat. Výsledky měření jsou k vidění v následujících grafech a tabulkách obsažených v příloze této bakalářské práce:

Verze ComplexMat:



Graf č.1 – Test průměrné rychlosti implementace ComplexMat

Verze cv::Mat:



Graf č.2 – Test průměrné rychlosti implementace cv::Mat

Podle získaných výsledků lze usoudit, že **verze cv::Mat je mírně pomalejší než ComplexMat**. To však v této fázi konverze není důležité, protože jejím hlavním úkolem je připravit projekt na následující konverze a odstranit redundantní třídy v kódu.

Z výsledků také plyne, že implementace používající knihovnu Fftw3 je v obou verzích projektu o něco výkonnější než OpenCV.

Směrodatné odchylky, vyznačené v grafu červenými chybovými úsečkami, nedosáhly většinou v průběhu testování výraznějších hodnot. Největší výkyvy byly zaznamenány při testu „bag“, kde se pohybovaly v okolí 55 ms.

Po dokončení všech testů došlo k odstranění postfixu „_Test“ z názvu testovacích proměnných a původní proměnné byly odstraněny z programu. Následný pokus o kompilaci pak odhalil všechny výskyty volání původních metod prostřednictvím kompilačních chyb. Všechna tato volání byla postupně také odstraněna.

Tím bylo úspěšně dosaženo dokončení konverze tříd ComplexMat a DynMem na cv::Mat.

Konverze na cv::UMat

Stejně jako při provádění konverze typu ComplexMat na cv::Mat byl kód předpřipraven pro práci podle zásad obecné metodiky, popsané v kapitole Návrh řešení (viz. výše).

Byly vytvořeny duplicitní proměnné typu cv::UMat, které byly inicializovány stejným způsobem jako původní proměnné typu cv::Mat, získané v předchozí konverzi.

Před začátkem vlastní implementace bylo však opět třeba zjistit jakým způsobem se bude lišit implementace cv::UMat od cv::Mat. Proto byly obě třídy před implementací konverze analyzovány v prostředí „implementačního hřiště“, tzn. v testovacím prostředí vytvořeném blízko místa začátku zpracování kódu, a předčasně ukončeném příkazem „return 0;“ (viz podkapitola Analýza kódu).

Analýzou obou tříd bylo zjištěno, že cv::UMat má podstatně menší rozsah definovaných funkcí než cv::Mat. Ať se jedná o manipulaci s daty nebo získávání informací o matici, cv::UMat má mnohem méně možností.

Naštěstí však cv::UMat obsahuje nástroje, kterými lze tyto nevýhody obejít. Za prvé, cv::UMat řadí svoje data ve stejném formátu jako cv::Mat. Nenastává tak problém, kterým se bylo třeba zabývat při konverzi z třídy ComplexMat.

Druhá věc která napomáhá potřebné konverzi je schopnost tříd cv::Mat a cv::UMat vytvořit hlavičku požadovaného typu na svoje vnitřní data pomocí vestavěných funkcí, konkrétně cv::UMat::getMat() a cv::Mat::getUMat(). Instance hlavičky typu cv::Mat je tedy schopna vytvořit hlavičku typu cv::UMat ukazující na stejná vnitřní data, a totéž platí v obráceném směru.

To znamená že rozsah možných operací nad danými daty, omezený použitým datovým typem, může být podle potřeby rozšířen dočasnou změnou na typ, který umožňuje širší rozsah operací.

Z tohoto důvodu byl v průběhu typové konverze projektu použit následující formát úprav:

- Všechny častěji používané proměnné jsou konvertovány na cv::UMat
- Tam kde je to možné jsou použity funkce definované v cv::UMat, jinak je typ dočasně konvertován na cv::Mat a operace je implementována pomocí metod této třídy
- Pokud by konverzí na typ cv::UMat došlo pouze k přidání zbytečné konverze v dané funkci, protože všechny operace uvnitř ní lze vykonat pouze pomocí metod cv::Mat, funkce i proměnné v ní budou ponechány bez konverze v typu cv::Mat
- To samé platí případech kdy konverze na cv::UMat z jakýchkoliv důvodů nedává praktický smysl

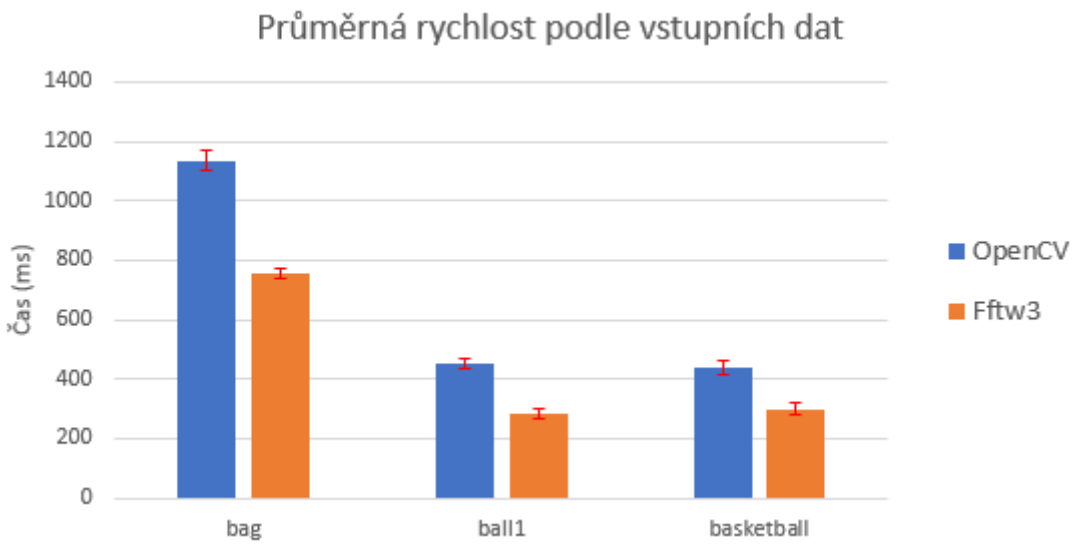
Za použití tohoto formátu úprav byla provedena konverze datových typů použitých v projektu. Během konverzí byly funkce také průběžně testovány na integritu dat a správnost výstupů. Funkční závislost programu pak byla přepnuta na duplicitní testovací proměnné, které byly cílem konvertovaných funkcí, za účelem otestování správného fungování programu.

Všechny tyto testy prošly bez problémů.

Dále prošla nová implementace porovnáním průměrné rychlosti zpracování jednotlivých snímků.

Formát testování byl zvolen stejný jako při testování konverze cv::Mat v předchozí podkapitole (viz. Konverze na cv::Mat).

S použitím tohoto formátu byla otestována verze programu používající datový typ cv::UMat. Výsledky měření jsou k vidění v následujícím grafu a tabulkách obsažených v příloze této bakalářské práce:



Graf č.3 – Test průměrné rychlosti implementace cv::UMat

Podle získaných výsledků lze usoudit, že **verze cv::UMat je mírně pomalejší než cv::Mat,** především v implementaci knihovny OpenCV, kde bylo provedeno nejvíce změn oproti předchozí verzi projektu.

Tento výsledek se dá vztáhnout k následujícím faktorům:

- Třída cv::UMat má oproti cv::Mat méně definovaných funkcí, které bylo možné použít k implementaci konverze. Proto funkce cv::UMat nebyly použity ve velkém rozsahu.
- Použití funkcí cv::UMat, které z definice pracují efektivnějším způsobem než funkce cv::Mat, mělo být hlavním faktorem zodpovědným za zrychlení programu
- Konverze projektu na datový typ cv::UMat vyžadovalo malé množství režijních nákladů v podobě opakovaného převádění typu maticových hlaviček

Z toho plyne, že použití typu cv::UMat nepřineslo zrychlení programu, přinejmenším ne v době zpracování bakalářské práce. Budoucí vývoj knihovny OpenCV však může otevřít nové možnosti použití této třídy.

Z výsledků testování opět plyne, že implementace používající knihovnu Fftw3 je v obou verzích projektu o něco výkonnější než OpenCV.

Směrodatné odchylky, vyznačené v grafu červenými chybovými úsečkami, nedosáhly v průběhu testování výraznějších hodnot.

Po dokončení konverze došlo k obvyklému začištění programu do konečné podoby této dané konverze.

Postfix „_Test“ byl odstraněn z názvu testovacích proměnných, a původní proměnné odstraněny z programu. Pokus o kompilaci pak odhalil všechny výskyty volání původních metod prostřednictvím kompilačních chyb. Všechna tato volání byla postupně odstraněna.

Tím bylo úspěšně dosaženo dokončení konverze tříd z cv::Mat na cv::UMat.

Konverze na implementaci Graph API

Tento stupeň konverze projektu se od ostatních liší tím, že zde nelze použít v plném rozsahu zásady obecné metodiky, popsané v kapitole Návrh řešení (viz. výše). Tento typ konverze se totiž z principu specifikace rozhraní týká pouze použitých funkcí, a nikoliv vnitřních proměnných jejichž obsah by bylo možné porovnávat s původními daty.

Místo toho je podstatou této konverze postup kódem ve směru zpracování kódu, a konverzí všech operací u kterých je to možné na implementaci pomocí rozhraní Graph API. Možností konverze je v tomto případě myšleno že daná operace určená ke konverzi musí být definována v rozhraní Graph API.

Počáteční analýza ukázala, že možnosti konverze jsou pro toto rozhraní velice omezené. Graph API je v době zpracování bakalářské stále ve vývoji, a chybí v něm definice mnoha výpočetně náročných operací, které jsou často používány v době běhu programu. Toto se týká zvláště operací nad maticovými daty, definovanými v souboru „matutil.h“, které jsou opakovaně volány ve smyčce while, starající se o tracking označené části obrazu v probíhajícím videu. Tyto funkce často pracují s maticemi komplexních dat, což operace definované v Graph API v naprosté většině nepodporují.

Nicméně, bylo nalezeno dostatečné množství funkcí vhodných ke konverzi, které ospravedlňují použití tohoto rozhraní. Jako součást implementace Graph API prošly konverzí především funkce upravující barevný formát obrazových matic, jejich typ, velikost a místa kódu, kde byly použity algebraické maticové výrazy.

Byl také učiněn pokus o použití Graph API k přenosu výpočtů na GPU, avšak i tento pokus nebyl příliš úspěšný. Zkušební test byl proveden vložením odkazu na jinou implementaci instrukcí napsaných v Graph API, napsanou v jazyce OpenCL. Tato implementace je součástí knihovny OpenCV a měla být schopná odesílat výpočetní operace explicitně ke zpracování v GPU.

Tato implementace Graph API však vykazuje závažné chyby znemožňující použití některých funkcí, které pracují správně ve výchozím implementaci. Testováním bylo navíc zjištěno, že funkce nové implementace jsou vždy alespoň 2 krát pomalejší než výchozí implementace Graph API.

Z toho důvodu nebyl přesun výpočetních operací na GPU proveden. Konverze funkcí do kontextu Graph API však zůstala zachována.

Tímto by tato část typové konverze projektu mohla být považována za dokončenou. V průběhu analýzy projektu byl však objeven alternativní způsob konverze, který podle informací získaných z článku na internetu umožňuje podstatně zvýšit efektivitu programu. [8]

Většina často používaných funkcí v souboru „matutil.h“ používá jako hlavní část svojí implementace iteraci dat pomocí smyčky for(), nativní pro jazyk C++. Pomocí této smyčky prochází daty v maticích za účelem čtení nebo zápisu dat podle svých potřeb.

Problém je že tento způsob procházení dat je výkonnostně neefektivní, a podle informací ve výše zmíněném článku existuje lepší algoritmus umožňující stejný výsledek.

Tento algoritmus je implementován ve funkci cv::Mat.forEach(), což je funkce implementovaná ve třídě cv::Mat, která je již používána v tomto projektu. Funkce cv::Mat.forEach() rozděluje práci iterace nad maticovými daty pomocí paralelních procesních vláken. Tím dochází k efektivnějšímu využití výkonové kapacity procesoru, a rychlejšímu zpracování dat.

Implementace cv::Mat.forEach() byla zkušebně implementována jako duplicitní volání v kódu, a její rychlost porovnána s původním voláním. Test ukázal, že nová implementace byla **2 až 3 krát rychlejší než původní provedení**. Podobného výsledku bylo dosaženo u každé další funkce souboru „matutil.h“, na které byla tato implementace použita.

Implementace cv::Mat.forEach() byla proto použita jako součást konečné verze tohoto stupně konverze projektu.

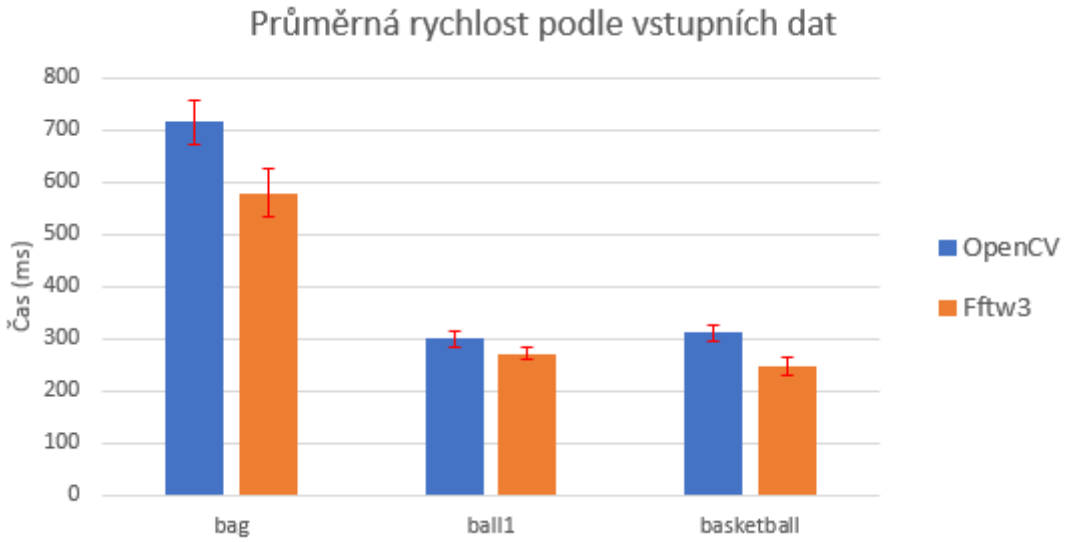
Všechny provedené změny byly průběžně testovány na integritu dat a správnost výstupů funkcí. V této fázi konverze projektu nebylo třeba přepínat funkční závislost programu na nových proměnných, nicméně byl stále proveden test na správné fungování programu.

Všechny tyto testy prošly bez problémů.

Dále prošla nová implementace porovnáním průměrné rychlosti zpracování jednotlivých snímků.

Formát testování byl zvolen stejný jako při testování konverze cv::Mat v předchozí podkapitole (viz. Konverze na cv::Mat).

S použitím tohoto formátu byla otestována verze programu používající implementaci rozhraní GAPI, bez konverze Fourierových transformací. Výsledky měření jsou k vidění v následujícím grafu a tabulkách obsažených v příloze této bakalářské práce:



Graf č.4 – Test průměrné rychlosti implementace Graph API (bez Fouriera)

Podle získaných výsledků lze usoudit, že **verze Graph API bez Fouriera je podstatně rychlejší než cv::UMat,** především v implementaci knihovny OpenCV, kde bylo provedeno nejvíce změn oproti předchozí verzi projektu.

Tato verze je dokonce rychlejší než výchozí verze programu použitého v této práci. Výkon při zpracování malých oblastí sledování je zhruba stejná jako ve verzi ComplexMat, ale **zpracování velkých oblastí sledování je zhruba o třetinu rychlejší.**

Tento výsledek se dá vztáhnout k následujícím faktorům:

- Funkce cv::Mat.forEach() dokáže procházet maticovými daty 2 až 3 krát rychleji než smyčka for() nativní pro jazyk C++
- Procházení dat pomocí funkce cv::Mat.forEach() byla použita u všech funkcí obsažených v souboru „matutil.h“. Tyto funkce jsou velmi často volány ve všech částech programu, a zvláště v hlavní trackovací smyčce.
- Konverze některých ostatních částí projektu pomocí rozhraní Graph API také přispěla ke zrychlení programu

Z toho plyne, že použití funkce přispělo velkou mírou ke zrychlení programu. Budoucí vývoj knihovny OpenCV může otevřít nové možnosti použití rozhraní Graph API, ale prozatím je funkce cv::Mat.forEach() jednoznačně lepší volbou.

Z výsledků testování opět plyne, že implementace používající knihovnu Fftw3 je v obou verzích projektu o něco výkonnější než OpenCV.

Rozdíl výkonností je však v tomto případě menší než u předchozích verzí programu, protože implementace Fftw3 závisí z větší části na funkcích své knihovny, a nedokáže tak efektivně využít funkcí upravených v rámci bakalářské práce.

Směrodatné odchylky, vyznačené v grafu červenými chybovými úsečkami, nedosáhly většinou v průběhu testování výraznějších hodnot. Největší výkyvy byly zaznamenány při testu „bag“, kde se pohybovaly v okolí 44 ms.

Začištění konečné verze této implementace nebylo tentokrát třeba provádět, protože všechny části konverze byly zaváděny s okamžitou účinností.

Tímto bylo úspěšně dosaženo dokončení první části implementace Graph API a cv::Mat.forEach().

Konverze Graph API pro Fourierovy transformace

Tato finální část bakalářské práce závisela z počátku především na analýze problému a navržení vhodného řešení. Neexistoval žádný předpis specifikující přesnou podobu požadovaného řešení, proto bylo potřeba provést počáteční výzkum a analýzu dostupných možností pro dokončení poslední části konverze projektu.

Úkolem bylo přesunout funkce Fourierových transformací, implementované různými způsoby pomocí knihoven OpenCV a Fftw3, do kontextu rozhraní Graph API.

Normálně by se jednalo o konverzi, kterou by bylo možné zahrnout v předchozí podkapitole. Problémem však je, že neexistuje žádná rovnocenná funkce z prostředí Graph API, kterou by bylo možné použít k implementaci Fourierovy transformace.

Proto bylo potřeba najít alternativní způsob, jak tuto implementaci definovat v kontextu Graph API, buď jako nově vloženou funkci, nebo jako kompozici existujících funkcí.

Po prozkoumání dokumentace OpenCV bylo zjištěno, že rozhraní Graph API počítá se situací, kdy je potřeba přidat novou funkci, a poskytuje nástroje k jejímu vytvoření. Tento nástroj se jmenuje Kernel API.

Jde o makro funkce definované v rozhraní Graph API, které umožňují deklarovat rozhraní nové funkce, a ve fázi kompilace balíčku instrukcí pak k těmto rozhraním přiřadit implementaci vhodnou pro daný problém. Tímto způsobem je možné použít vhodnou implementace z různých programovacích jazyků, což otevírá nové možnosti optimalizace.

Dostupná dokumentace Kernel API byla prozkoumána a její implementace zkušebně aplikována na funkce Fourierových transformací definovaných pomocí knihovny OpenCV.

Nejprve bylo definováno rozhraní nové funkce, typ vstupních argumentů a metadata popisující výstupní matici. Pro implementaci tohoto rozhraní byla použita původní funkce Fourierovy transformace, jménem cv::dft(), obsažená v knihovně OpenCV.

Tímto byla funkce připravena k použití v programu. Původní funkce však byla zachována v kódu pod stejným jménem a postfixem „_cpu“.

Po dokončení konverze bylo možno tuto novou funkci zavolat při sestavování balíčku instrukcí v kontextu Graph API. Definice nové funkce pak byla vložena jako argument navíc ve fázi kompilace balíčku. Při testování měla funkce stejný výstup jako původní implementace.

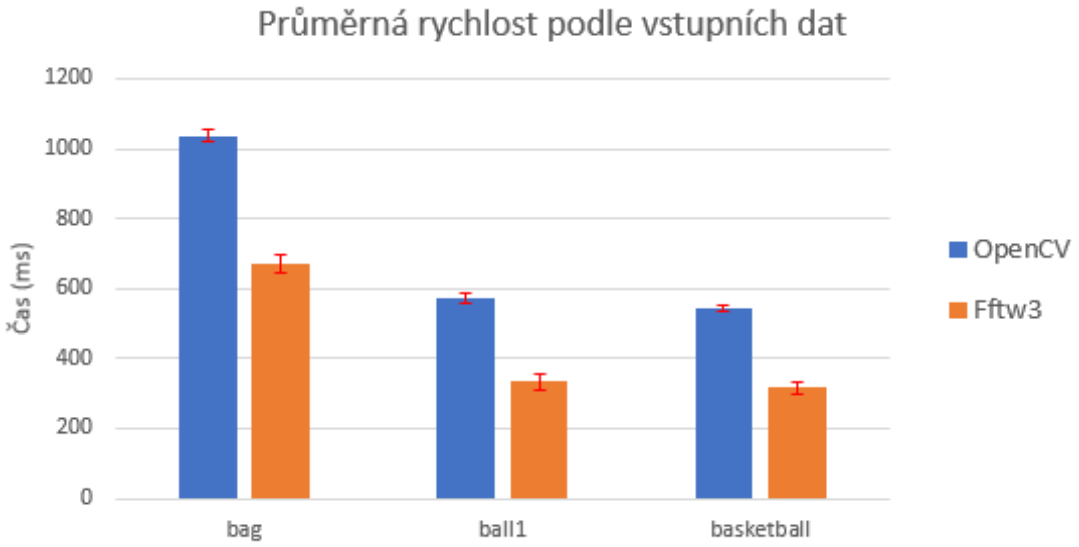
Podobným způsobem pak byly konvertovány i ostatní funkce z knihovny OpenCV a Fftw3, a otestovány na správnost výstupu a integrity dat.

Všechny tyto testy prošly bez problémů.

Dále prošla nová implementace porovnáním průměrné rychlosti zpracování jednotlivých snímků.

Formát testování byl zvolen stejný jako při testování konverze cv::Mat v předchozí podkapitole (viz. Konverze na cv::Mat).

S použitím tohoto formátu byla otestována verze programu používající implementaci rozhraní GAPI, tentokrát s konverzí Fourierových transformací. Výsledky měření jsou k vidění v následujícím grafu a tabulkách obsažených v příloze této bakalářské práce:



Graf č.5 – Test průměrné rychlosti implementace Graph API (s Fourierem)

Podle získaných výsledků lze usoudit, že **verze Graph API s Fourierem je podstatně pomalejší než Graph API bez Fouriera.** Toto platí téměř výhradně pro implementaci knihovny OpenCV, s mírným zpomalením pro řešení používající Fftw3.

Důvodem tohoto zpomalení byly následující faktory:

- Při použití funkce cv::dft() se vyskytla komplikace, znemožňující obvyklý způsob jejího použití v kontextu Kernel API
- Kernel API v rámci svojí specifikace vyžaduje, aby adresa vnitřního ukazatele výstupní matice, zadané jako argument nové funkce, zůstala stejná jako před zpracováním.
- Funkce cv::dft() mění adresu vnitřního ukazatele pro matici kterou dostane jako argument, a proto ji nelze použít přímo na výstupní matici zadanou jako argument Kernel API funkce.
- Proto byl použit mezikrok, který zahrnuje použití cv::dft() na nově vytvořenou matici, a následné překopírování dat z nové matice do výstupní.
- K implementaci mezikroku byla použita funkce cv::Mat.forEach(), ale vzniklá výpočetní zátěž i přesto způsobila podstatné zpomalení nově definované funkce.

Z toho plyne, že dokud bude Kernel API vyžadovat neměnnost vnitřního ukazatele výstupu, nebo dokud funkce cv::dft() nepřestane tuto podmínku porušovat, výsledná konverze Graph API bude vždy výrazně pomalejší, a z toho důvodu by neměla být použita.

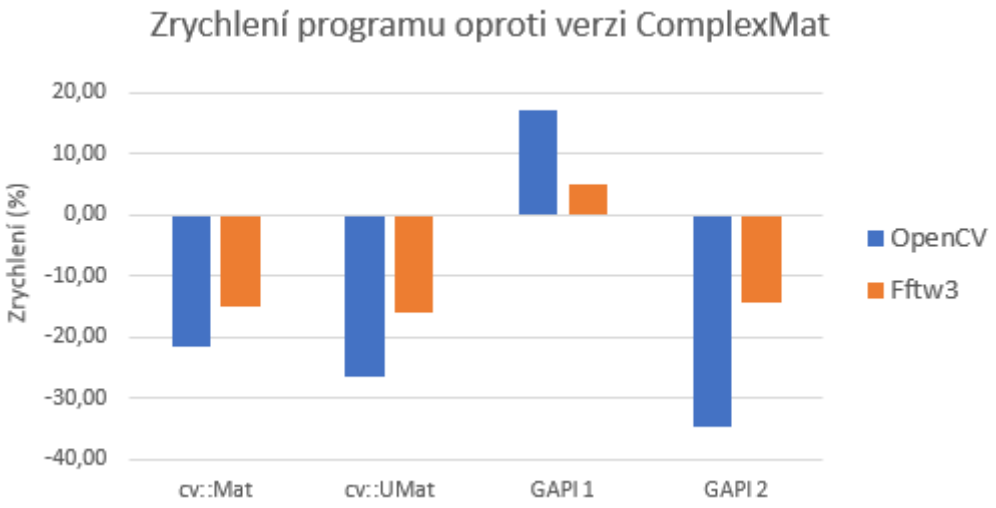
Implementace používající knihovnu Fftw3 zaznamenala velmi mírné zpomalení v důsledku režie použití Kernel API, ale jinak je stejně výkonná jako v předchozí verzi konverze Graph API. A tím víc je také rychlejší než implementace OpenCV.

Směrodatné odchylky, vyznačené v grafu červenými chybovými úsečkami, nedosáhly v průběhu testování výraznějších hodnot.

Volání původních funkcí bylo po dokončení testování nahrazeno novou implementací ve všech místech v kódu, se zachováním definice původní efektivnější funkce. Tuto funkci je možno kdykoliv použít místo současného řešení.

V této fázi bylo úspěšně dosaženo dokončení implementace Graph API v celém rozsahu.

Přehled vlivu jednotlivých konverzí na průměrnou rychlost programu oproti počáteční verzi je k vidění v následujícím grafu:



Graf č.6 – Zrychlení programu oproti původní verzi po aplikaci konverzí

Provedené úpravy programu jsou k vidění v repositáři GitHub [6]. Dokončené stupně konverze mají v úložišti definované následující tagy, pod kterými je možné je najít:

- Kcf-original (commit „25ae81f“)
- Kcf-Mat_Conv (commit „0eeba28“)
- Kcf-UMat-Conv (commit „535079b“)
- Kcf-GAPI1-Conv (commit „51f5d02“)
- Kcf-GAPI2-Conv (současná verze)

5) Závěr

V rámci bakalářské práce jsem se do detailů seznámil se strukturou upravovaného projektu, což mi umožnilo navrhnout způsob jeho zpracování. Vytvořil jsem plán konverze projektu pro použití s datovými typy cv::Mat, cv::UMat a rozhraním Graph API.

Podle tohoto plánu jsem potom úspěšně vytvořil a otestoval implementace těchto konverzi podle kritérií funkčnosti. Z hlediska zrychlení programu jsem dosáhl úspěchu, zvláště zřetelného ve fázi konverze Graph API bez Fouriera. Ostatní části konverze projektu nebylo možné zrychlit z důvodu omezení rozhraní GraphAPI a datového typu cv::UMat, které jsou oba stále ve vývoji.

Kód projektu nyní obsahuje nejoptimálnější verzi z daných konverzí, společně s fungujícím řešením Fourierových transformací, implementovaných pomocí Graph API. Toto řešení bude možné v budoucnu použít k další optimalizaci, až dojde k vydání nové verze knihovny OpenCV.

Průběh zpracování bakalářské práce byl popsán v této zprávě, spolu s výsledky testování a porovnáním rychlostí konvertovaných verzí programu.

Zpracování této bakalářské práce se pro mě stala cennou zkušeností, která se bude hodit, pokud budu někdy v budoucnosti pracovat s jinými projekty pracujícími na základě konceptu počítačového vidění.

Reference a odkazy

[1] Tomáš Vojíř. „Short-Term Visual ObjectTracking in Real-Time“. disertaceVojir [online]. [cit. 2020-05-17]. Dostupné z: http://cyber.felk.cvut.cz/teaching/radaUIB/disertaceVojir.pdf
[2] OpenCV documentation. OpenCV [online]. [cit. 2020-01-10]. Dostupné z: https://docs.opencv.org/4.2.0
[3] Graph API. OpenCV: Graph API [online]. [cit. 2020-05-17]. Dostupné z: https://docs.opencv.org/master/d0/d1e/gapi.html
[4] Kernelized Correlation Filter tracker. GitHub [online]. Dostupné z: https://github.com/CTU-IIG/kcf
[5] G-API Kernel API. OpenCV: Kernel API [online]. [cit. 2020-05-14]. Dostupné z: https://docs.opencv.org/master/d0/d25/gapi_kernel_api.html
[6] Github tracker personal repository. Github [online]. [cit. 2020-01-10]. Dostupné z: https://github.com/oraveja1/kcf
[7] Visual object mapping Datasets. VOT2016 Challenge|Dataset [online]. [cit. 2020-05-14]. Dostupné z: https://www.votchallenge.net/vot2016/dataset.html
[8] Parallel Pixel Access in OpenCV using forEach. Learn OpenCV [online]. [cit. 2020-05-14]. Dostupné z: https://www.learnopencv.com/parallel-pixel-access-in-opencv-using-foreach/
[9] João F. Henriques, Rui Caseiro, Pedro Martins, Jorge Batista, “High-Speed Tracking with Kernelized Correlation Filters“, IEEE Transactions on Pattern Analysis and Machine Intelligence, 2015

6) Přílohy

Výsledky testů průměrných rychlostí zpracování snímku

Verze ComplexMat:

Číslo testu	1	2	3	4	5	6	7	8	9	10
bag [ms]	1003,71	1037,36	917,13	997,00	1012,54	1002,67	880,49	925,51	972,52	954,16
ball1 [ms]	227,93	224,75	240,61	255,72	267,55	240,98	255,34	256,05	272,33	255,50
basketball [ms]	360,82	399,93	350,42	384,50	362,01	381,12	387,32	385,69	386,95	409,00

Tabulka č. 1 – Test rychlosti s knihovnou OpenCV, verze ComplexMat

Číslo testu	1	2	3	4	5	6	7	8	9	10
bag [ms]	821,96	672,70	620,66	572,93	682,22	710,46	662,25	670,02	655,47	655,84
ball1 [ms]	205,54	211,54	219,88	235,15	218,72	235,30	220,06	218,69	235,58	233,29
basketball [ms]	241,06	255,63	248,76	240,64	267,81	267,30	265,91	262,56	240,19	280,26

Tabulka č. 2 – Test rychlosti s knihovnou Fftw3, verze ComplexMat

Verze cv::Mat:

Číslo testu	1	2	3	4	5	6	7	8	9	10
bag [ms]	1250,78	1166,67	1084,67	1104,47	1183,75	1103,87	1180,43	1103,86	1083,81	1129,27
ball1 [ms]	377,50	383,85	395,70	398,79	382,43	360,27	345,29	350,44	328,78	323,46
basketball [ms]	449,94	440,26	447,82	420,84	449,70	435,22	421,11	448,03	476,10	453,13

Tabulka č.3 – Test rychlosti s knihovnou OpenCV, verze cv::Mat

Číslo testu	1	2	3	4	5	6	7	8	9	10
bag [ms]	755,75	748,00	740,89	756,80	743,31	739,48	713,36	769,22	766,83	764,56
ball1 [ms]	286,71	287,12	300,81	289,88	303,66	309,4	286,83	280,14	304,02	316,5
basketball [ms]	290,03	277,98	284,73	317,40	279,76	250,90	288,74	284,50	257,40	253,31

Tabulka č.4 – Test rychlosti s knihovnou Fftw3, verze cv::Mat

Verze cv::UMat:

Číslo testu	1	2	3	4	5	6	7	8	9	10
bag [ms]	1155,53	1160,89	1113,35	1199,79	1147,34	1145,38	1114,60	1087,81	1096,81	1143,39
ball1 [ms]	479,41	445,39	421,08	452,73	478,77	444,58	461,12	449,83	431,58	438,40
basketball [ms]	436,34	463,48	463,21	422,54	395,64	414,03	453,14	439,41	422,02	464,24

Tabulka č.5 – Test rychlosti s knihovnou OpenCV, verze cv::UMat

Číslo testu	1	2	3	4	5	6	7	8	9	10
bag [ms]	730,93	757,25	719,96	750,40	742,33	753,31	771,73	762,41	764,63	778,99
ball1 [ms]	276,35	264,94	314,76	276,34	305,03	292,25	291,3	286,21	267,96	275,72
basketball [ms]	309,48	310,82	295,21	284,63	324,59	317,63	288,42	256,66	291,99	313,83

Tabulka č.6 – Test rychlosti s knihovnou Fftw3, verze cv::UMat

Verze Graph API - bez Fouriera:

Číslo testu	1	2	3	4	5	6	7	8	9	10
bag [ms]	770,14	754,98	674,38	702,08	650,57	764,99	704,28	745,37	654,07	737,05
ball1 [ms]	283,67	312,52	313,08	299,54	298,74	285,29	292,62	272,77	334,08	299,23
basketball [ms]	315,81	311,01	287,06	327,56	324,62	311,43	286,79	302,60	328,70	311,30

Tabulka č.7 – Test rychlosti s knihovnou OpenCV, verze Graph API (bez Fouriera)

Číslo testu	1	2	3	4	5	6	7	8	9	10
bag [ms]	624,08	600,41	554,27	531,85	536,91	520,64	539,16	640,66	651,64	587,28
ball1 [ms]	275,78	251,68	270,55	256,65	271,68	263,99	284	276,4	273,54	292,15
basketball [ms]	255,49	247,02	251,20	230,45	241,93	228,64	286,94	255,18	244,90	229,74

Tabulka č.8 – Test rychlosti s knihovnou Fftw3, verze Graph API (bez Fouriera)

Verze Graph API – s Fourierem:

Číslo testu	1	2	3	4	5	6	7	8	9	10
bag [ms]	1076,29	1014,41	1032,83	1023,61	1051,84	1032,63	1039,14	1040,25	1023,18	1044,00
ball1 [ms]	584,62	566,84	540,93	565,32	595,26	570,85	576,98	593,19	578,15	567,16
basketball [ms]	554,19	535,60	551,27	532,64	548,91	553,65	548,33	535,54	528,96	543,58

Tabulka č.9 – Test rychlosti s knihovnou OpenCV, verze Graph API (s Fourierem)

Číslo testu	1	2	3	4	5	6	7	8	9	10
bag [ms]	669,14	647,24	728,51	674,65	672,42	631,88	670,19	683,03	672,69	644,63
ball1 [ms]	315,64	306,13	317,79	377,62	332,16	373,72	330,98	331,69	345,94	303,17
basketball [ms]	315,67	333,97	311,84	306,12	358,71	315,12	299,61	312,57	320,52	296,77

Tabulka č.10 – Test rychlosti s knihovnou Fftw3, verze Graph API (s Fourierem)