



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

NÁVRH VESTAVANÉHO SYSTÉMU INTELIGENTNÉHO VIDĚNÍ NA PLATFORMĚ NVIDIA

EMBEDDED VISION SYSTEM DESIGN ON NVIDIA PLATFORM

DIPLOMOVÁ PRÁCE

MASTER'S THESIS

AUTOR PRÁCE

AUTHOR

Bc. Filip Krivoklatský

VEDOUCÍ PRÁCE

SUPERVISOR

doc. Ing. Zdeněk Bradáč, Ph.D.

BRNO 2019

Diplomová práce

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

Ústav automatizace a měřicí techniky

Student: Bc. Filip Krivoklatský

ID: 173684

Ročník: 2

Akademický rok: 2018/19

NÁZEV TÉMATU:

Návrh vestavaného systému inteligentního vidění na platformě NVIDIA

POKYNY PRO VYPRACOVÁNÍ:

1. Proveďte literární řešení a nastiňte problematiku projektu.
2. Navrhněte koncepci systému vidění na platformě NVIDIA s operačním systémem Linux. Navrhněte koncepci systému a připojení nezbytných periférií.
3. Navrhněte koncepci programového vybavení a nezbytných programových bloků, zaměřte se na multivláknové zpracování dat a využití technologie CUDA.
4. Zrealizujte elektronický systém a oživte jej. Zprovozněte potřebný tool-chain.
5. Implementujte programové vybavení. Vytvořte SDK knihovnu pro vzdálené ovládání systému z PC přes rozhraní Ethernet.
6. Porovnejte toto řešení s řešením založeným na klasickém PC a zhodnoťte výsledky.

DOPORUČENÁ LITERATURA:

Pavel Herout: Učebnice jazyka C, KOPP, 2004, IV. přepracované vydání, ISBN 80-7232-220-6

Dle pokynů vedoucího práce.

Termín zadání: 4.2.2019

Termín odevzdání: 13.5.2019

Vedoucí práce: doc. Ing. Zdeněk Bradáč, Ph.D.

Konzultant:

doc. Ing. Václav Jirsík, CSc.
předseda oborové rady

UPOZORNĚNÍ:

Autor diplomové práce nesmí při vytváření diplomové práce porušit autorská práva třetích osob, zejména nesmí zasahovat nedovoleným způsobem do cizích autorských práv osobnostních a musí si být plně vědom následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., včetně možných trestněprávních důsledků vyplývajících z ustanovení části druhé, hlavy VI. díl 4 Trestního zákoníku č.40/2009 Sb.

Abstrakt

Táto diplomová práca sa zaoberá návrhom vstavaného systému počítačového videnia na platforme Nvidia a prenosom existujúcej aplikácie na 3D detekciu objektov z platformy Windows do navrhnutého systému s operačným systémom Linux. Práca rieši návrh komunikačného rozhrania pre ovládanie systému a prenos obrazu z kamery po lokálnej sieti s využitím video kompresie. Pre zrýchlenie behu detekčného algoritmu sú výpočtovo náročné funkcie detekčného programu presunuté na GPU s využitím technológie CUDA. Na ovládanie celého systému vzdialene je navrhnutá ovládacia aplikácia s grafickým rozhraním pre platformu Windows.

Kľúčové slová

počítačové videnie, Nvidia Jetson TX2, CUDA, OpenCV, FLIR, kompresia videa, H.264, H.265, FFmpeg, GStreamer, TCP, UDP

Abstract

This diploma thesis deals with design of embedded computer vision system and transfer of existing computer vision application for 3D object detection from Windows OS to designed embedded system with Linux OS. Thesis focuses on design of communication interface for system control and camera video transfer through local network with video compression. Then, detection algorithm is enhanced by transferring computationally expensive functions to GPU using CUDA technology. Finally, a user application with graphical interface is designed for system control on Windows platform.

Keywords

computer vision, Nvidia Jetson TX2, CUDA, OpenCV, FLIR, video compression, H.264, H.265, FFmpeg, GStreamer, TCP, UDP

Bibliografická citácia:

KRIVOKLATSKÝ, Filip. Návrh vestavaného systému inteligentného vidění na platformě NVIDIA [online]. Brno, 2019 [cit. 2019-05-09]. Dostupné z: <https://www.vutbr.cz/studenti/zav-prace/detail/119258>. Diplomová práce. Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav automatizace a měřicí techniky. Vedoucí práce Zdeněk Bradáč.

Prehlásenie

„Prehlasujem, že svoju záverečnú prácu na tému *Návrh vestavaného systému inteligentného vidění na platformě NVIDIA* som vypracoval samostatne pod vedením vedúceho záverečnej práce a s použitím odbornej literatúry a ďalších informačných zdrojov, ktoré sú všetky citované a uvedené v zozname literatúry na konci práce.

Ako autor uvedenej záverečnej práce ďalej prehlasujem, že v súvislosti s vytvorením tejto práce som neporušil autorské práva tretích osôb, najmä som nezasiahol nedovoleným spôsobom do cudzích autorských práv osobnostných a som si plne vedomý následkov porušenia ustanovení § 11 a nasledujúcich autorského zákona č. 121/2000 Sb., vrátane možných trestnoprávných dôsledkov vyplývajúcich z ustanovenia časti druhej hlavy VI. diel 4 Trestného zákonníku č. 40/2009 Sb.

V Brne dňa

.....

podpis autora

Pod'akovanie

Ďakujem vedúcemu diplomovej práce doc. Ing. Zdeněkovi Bradáčovi, Ph.D za účinnú metodickú, pedagogickú a odbornú pomoc a ďalšie cenné rady pri spracovaní mojej záverečnej práce.

V Brne dňa

.....

podpis autora

Obsah

Úvod	1
1 Počítačové videnie	2
1.1 Systémy počítačového videnia	2
1.1.1 Systémy inteligentného videnia	2
1.2 Priemyselné systémy.....	3
1.2.1 Príklady použitia priemyselných systémov	3
1.3 Detekcia 3D objektov v priestore	5
1.3.1 Algoritmus detekcie	5
1.4 OpenCV	9
1.4.1 OpenCV a CUDA	9
2 Platforma NVIDIA	10
2.1 Nvidia CUDA	11
2.1.1 Programovací model CUDA.....	11
2.1.2 Nvidia CUDA Toolkit.....	15
2.2 NVIDIA JetPack	15
3 Komunikácia a prenos dát.....	16
3.1 Komunikačné protokoly.....	16
3.1.1 TCP protokol	16
3.1.2 UDP protokol	17
3.1.3 Knižnica Asio	17
3.2 Prenos živého obrazu.....	18
3.2.1 Kompresia dát.....	18
3.2.2 FFmpeg.....	20
3.2.3 GStreamer	20
4 Návrh systému	22
4.1 Požiadavky na systém.....	22
4.2 Bloková schéma systému.....	22
4.3 Výber platformy	23
4.3.1 NVIDIA Jetson TX2 (TX2i).....	23
4.3.2 Vývojový kit pre Nvidia Jetson TX2	24
4.3.3 Nosná doska	24
4.4 Výber kamery	25
4.4.1 Výber rozhrania.....	25
4.4.2 Výber kamery	28

4.5	Komunikácia so zariadením	29
4.6	Výsledná bloková schéma	29
5	Návrh softvéru a implementácia	31
5.1	Oživovanie vývojovej dosky Nvidia Jetson TX2	32
5.2	Komunikačná knižnica.....	32
5.2.1	TCP Server	33
5.2.2	TCP Klient	33
5.2.3	Prenos správ	33
5.2.4	Prenos súborov	34
5.2.5	Aplikačné rozhranie knižnice.....	36
5.3	Prenos obrazu	37
5.3.1	Odosielanie streamu	37
5.3.2	Príjem streamu	39
5.3.3	Latencia prenosu.....	39
5.4	Hlavná aplikácia	40
5.4.1	Preklad aplikácie v prostredí Linux.....	40
5.4.2	Popis aplikácie	40
5.4.3	Nvidia CUDA.....	42
5.4.4	Oživovanie kamery	52
5.5	Ovládacia aplikácia.....	53
5.5.1	Qt	53
5.5.2	Popis aplikácie	53
6	Dosiahnuté výsledky	57
6.1	Porovnanie časov detekčného algoritmu	57
6.1.1	Výsledky modulu Jetson TX2.....	57
6.1.2	Výsledky testovacieho počítača.....	59
	Záver	62
	Literatúra	64
	Zoznam skratiek.....	67
	Zoznam príloh	68

Zoznam obrázkov

Obr. 1: Kontrola vady fľaše počítačovým videním [29]	2
Obr. 2: Detail oblasti sledovanej kamerou [28]	3
Obr. 3: Systém ukladania objektov na dopravník [28]	3
Obr. 4: Kontrola plnenia fliaš s vodou [28]	4
Obr. 5: Kontrola skrutkového spoja [28]	4
Obr. 6: Testovací 3D objekt	5
Obr. 7: Vytvorenie šablóny z 3D objektu.....	6
Obr. 8: Zobrazenie nájdeného objektu detektorom.....	6
Obr. 9: Blokový diagram programu detekcie	7
Obr. 10: Predspracovanie obrazu programom 3D detekcie.....	8
Obr. 11: Porovnanie CPU a GPU pri operáciách s desatinnou čiarkou [32].....	12
Obr. 12: Porovnanie štruktúry CPU a GPU [32].....	12
Obr. 13: Rozdelenie CUDA programu na GPU	13
Obr. 14: TCP Handshake [20]	16
Obr. 15: Porovnanie kodekov H.264 a H.265 (Full HD, 400kb/s) [25]	19
Obr. 16: Bloková schéma systému.....	22
Obr. 17: Modul NVIDIA Jetson TX2 [4].....	23
Obr. 18: Vývojový kit Nvidia Jetson TX2 [6]	24
Obr. 19: Nosná doska Orbitty Carrier [39]	25
Obr. 20: Kamera FLIR BFS-PGE-31S4C-C [30]	28
Obr. 21: Výsledná bloková schéma systému.....	29
Obr. 22: Štruktúra programových blokov.....	31
Obr. 23: Vývojový diagram hlavnej aplikácie.....	41
Obr. 24: Výpis z programu CMake	43
Obr. 25: Vývojové diagramy funkcie hľadania kandidátov.....	48
Obr. 26: Vývojové diagramy funkcie upresnenia pozície	51
Obr. 27: Výzva klientskej aplikácie na pripojenie	53
Obr. 28: Hlavné okno ovládacej klientskej aplikácie.....	54
Obr. 29: Hlavné okno ovládacej klientskej aplikácie so zapnutým streamom	55
Obr. 30: Hlavné okno ovládacej klientskej aplikácie so zapnutou detekciou	56
Obr. 31: Záložky aplikácie Preprocessing a Detection.....	56

Zoznam tabuliek

Tab. 1: Prehľad zariadení platformy Nvidia	10
Tab. 2: Dátová reprezentácia štruktúry po zabalení cez msgpack.....	36
Tab. 3: Porovnanie rýchlostí predspracovania.....	46
Tab. 4: Zoznam módov procesora modulu Jetson TX2 [40]	46
Tab. 5: Namerané časy detekčného programu na Jetson TX2	57
Tab. 6: Porovnanie parametrov grafických jednotiek PC a modulu Jetson TX2	60
Tab. 7: Namerané časy detekčného programu na testovanom PC.....	60

Zoznam výpisov kódu

Výpis 1: Príklad použitia CUDA v jazyku C	14
Výpis 2: Formát posielanej správy cez TCP	34
Výpis 3: Ukážka hlavičky pri posielaní súboru cez TCP	34
Výpis 4: Príklad užívateľsky definovanej štruktúry pre msgpack	35
Výpis 5: Výstupná GStreamer pipeline	38
Výpis 6: Určenie ukazateľa na požadovanú šablónu a ošetrovanie rozsahu	49

Zoznam grafov

Graf 1: Čas funkcie hľadania kandidátov v závislosti od počtu šablón objektu	58
Graf 2: Čas funkcie upresnenia pozície v závislosti od počtu šablón objektu.....	58
Graf 3: Celkový čas detekčného algoritmu v závislosti od počtu šablón objektu.....	59
Graf 4: Čas funkcie hľadania kandidátov v závislosti od počtu šablón objektu	60
Graf 5: Čas funkcie upresnenia pozície v závislosti od počtu šablón objektu.....	61
Graf 6: Celkový čas detekčného algoritmu v závislosti od počtu šablón objektu.....	61

ÚVOD

Počítačové videnie je oblasť, ktorá v posledných rokoch zaznamenáva veľký rozmach. Systémy využívajúce počítačové videnie sa s výhodou nasádzajú do priemyselnej výroby na miesta, ktoré musel v minulosti robiť človek a zefektívňuje sa tak celý proces výroby alebo iných operácií v priemysle. Takýmto systémom sa budeme venovať aj my v tejto práci.

Práca sa bude zaoberať návrhom vstaveného systému počítačového videnia. Úlohou práce bude preniesť existujúcu aplikáciu počítačového videnia z platformy Windows na vstavané zariadenie platformy Nvidia s operačným systémom Linux. Navrhnutý systém bude možné ovládať zo vzdialeného počítača pomocou vytvorenej aplikácie s použitím vytvorenej komunikačnej knižnice.

Aplikácia počítačového videnia, z ktorej budeme vychádzať v tejto práci, bola poskytnutá spolupracujúcou firmou venujúcou sa počítačovému videniu, a je jej vlastníctvom. V práci bude popísaný len základný princíp činnosti aplikácie a algoritmus detekcie. Táto aplikácia sa používa na detekciu 3D objektov v priemyselných podmienkach, pričom dokáže z 2D obrazu jednej kamery určiť polohu známeho 3D objektu s presnosťou na niekoľko milimetrov. Výstupom aplikácie je poloha nájdeného objektu v 3D priestore a percentuálna zhoda so zadaným objektom.

V práci budú v úvodných kapitolách popísané teoretické poznatky, ktoré budeme v práci využívať k návrhu a implementácii systému. Popísané budú existujúce priemyselné systémy počítačového videnia, platforma Nvidia a zariadenia, ktoré táto spoločnosť ponúka. V práci sa tiež budeme venovať problematike prenosu obrazu po lokálnej sieti cez protokol IP a jeho kompresii.

V praktických kapitolách sa budeme venovať návrhu systému počítačového videnia a vyberieme vhodné komponenty. Ďalšia kapitola sa bude venovať návrhu softvéru pre celý systém, kde si najskôr stanovíme štruktúru programových blokov, ktoré budeme vytvárať. Následne popíšeme vytvorenie komunikačnej knižnice a venovať sa budeme využitiu technológie CUDA pre zrýchlenie algoritmu detekcie 3D objektov.

V poslednej kapitole uvedieme dosiahnuté výsledky práce a porovnáme vytvorený vstavaný systém s klasickým počítačovým systémom. Výsledkom práce budú dve vytvorené aplikácie, z ktorých jedna bude bežať na navrhnutom systéme a druhá aplikácia bude užívateľovi poskytovať grafické rozhranie na ovládanie a správu systému zo vzdialeného počítača.

1 POČÍTAČOVÉ VIDENIE

Počítačové videnie je vedecká oblasť, ktorá rieši problém spracovania a pochopenia digitálnych obrázkov alebo videí. Z pohľadu priemyslu sa jedná o systémy, ktoré sa usilujú o zvládnutie úloh, ktoré človek zvláda svojimi zrakovými zmyslami. Počítačové videnie rieši metódy získavania, spracovania, analýzy a pochopenia digitálnych obrázkov a extrakciu dát z reálneho sveta na získanie numerickej alebo symbolickej informácie. Pochopenie v tomto kontexte znamená transformáciu vizuálnych obrázkov na deskripty reálneho sveta, ktoré dokážu interagovať s ďalšími myšlienkovými procesmi a vykonať požadovanú akciu [12]. Oblasť počítačového videnia zažíva v posledných rokoch svoj veľký rozmach. K jej rozvoju došlo hlavne kvôli nárastu výkonu počítačov a vytvoreniu nástrojov počítačového videnia.

1.1 Systémy počítačového videnia

Systémy počítačového videnia sa začínajú nasadzovať najmä v priemysle, kde slúžia pri výrobe a kontrole výrobného procesu, ale oblastí použitia je oveľa viac. Pomocou technológií počítačového videnia sa stroje nasádzajú na pracovné miesta, ktoré v minulosti nebolo možné nahradiť, najmä pri kontrole výrobného procesu.

Tieto systémy sa používajú na veľké spektrum problémov. Na obrázku vpravo môžeme vidieť príklad použitia počítačového videnia na kontrolu vady sklenenej fľaše. Uvedieme si aj niektoré ďalšie príklady použitia:

- Klasifikácia objektov,
- Detekcia objektov/defektov,
- Meranie rozmerov,
- 3D meranie rozmerov,
- Sledovanie pohybu,
- OCR – rozpoznávanie textu z obrazu.



Obr. 1: Kontrola vady fľaše počítačovým videním [29]

1.1.1 Systémy inteligentného videnia

Týmto názvom sú označované systémy, ktoré obsahujú určitú inteligenciu, ktorú využívajú pri rozhodovaní. Väčšinou sa jedná o naučené neurónové siete, ktoré tak dokážu reagovať na vopred naučené situácie, ale zároveň sa dokážu učiť zo skúseností. Tieto systémy vznikli použitím počítačového videnia a umelej inteligencie. Ako systémy inteligentného videnia sú však označované aj klasické systémy počítačového videnia. My sa v práci budeme venovať systémom videnia bez použitia umelej inteligencie.

1.2 Priemyselné systémy

Priemysel je jedným z odvetví, kde sa vo veľkej miere rozširujú systémy s využitím počítačového videnia. V priemysle sa tieto systémy používajú hlavne pri týchto aplikáciách:

- Kontrola kvality súčiastok,
- Snímanie kódov – identifikácia,
- Rozpoznávanie objektov,
- Počítanie kusov v balení,
- Meranie výrobku: šírka, dĺžka, trhliny a pod.,
- Určovanie pozície objektov, polohy produktov (navádzanie robota).

Kvalitu ľudského videnia je ťažké napodobniť. Kamery vytvorené človekom ani zďaleka nedosahujú tieto kvality, či už v rozlíšení, ale najmä v dynamickom rozsahu oka. Ľudia však nie sú stroje a pre operátorov výrobných liniek je veľmi náročné udržať pozornosť a kontrolovať kvalitu súčiastok počas celej pracovnej doby. Človek sa už po niekoľkých minútach nedokáže plne sústrediť na prácu. Priemyselné systémy videnia takéto úlohy zvládajú nepretržite a vo väčšine prípadov fungujú podstatne rýchlejšie ako človek. Vlastnosti týchto systémov, ako: presnosť, rýchlosť, robustnosť a opakovateľnosť zaradujeme medzi hlavné výhody použitia priemyselných systémov. [28]

1.2.1 Príklady použitia priemyselných systémov

Teraz si uvedieme niekoľko príkladov použitia počítačového videnia v priemysle na zefektívnenie a zrýchlenie výrobného procesu.

1.2.1.1 Určovanie polohy objektov pre uchopenie robotom

Na nasledujúcich obrázkoch je zobrazený proces ukladania objektov (kľúčov) na dopravníkový pás. Hlavnou časťou systému je kamera, ktorá sleduje oblasť



Obr. 3: Systém ukladania objektov na dopravník [28]



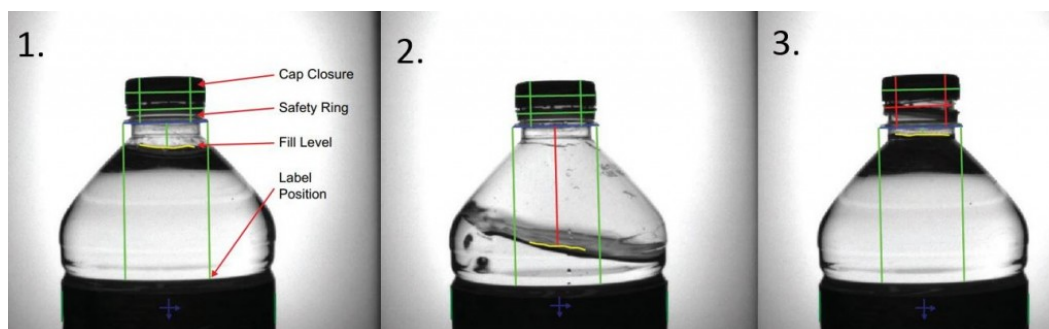
Obr. 2: Detail oblasti sledovanej kamerou [28]

vyznačenú na druhom obrázku a zo snímok určí polohu jednotlivých objektov. Informácia o polohe je potom predaná robotovi, ktorý uchopí daný objekt. V tejto aplikácii sa počítačové videnie využíva na 2D detekciu objektov z obrazu kamery.

1.2.1.2 Kontrola plnenia fliaš s vodou

Na nasledujúcom obrázku je znázornený proces kontroly fl'aše s vodou. Kontroluje sa správne množstvo vody, bezpečnostný krúžok a uzavretie fl'aše. Popis obrázkov:

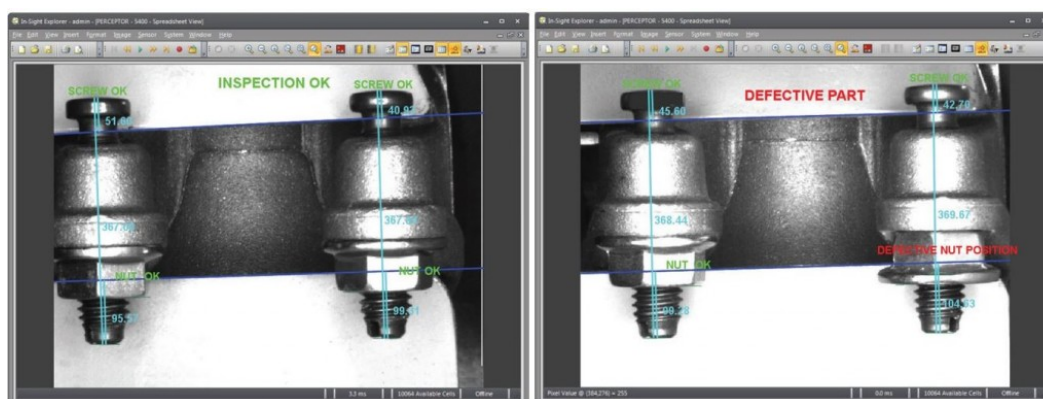
1. Úspešná kontrola – správne uzavretá a naplnená fl'aša.
2. Neúspešná kontrola – detegovaná nízka hladina vody.
3. Neúspešná kontrola – detegované nesprávne osadenie uzáveru.



Obr. 4: Kontrola plnenia fliaš s vodou [28]

1.2.1.3 Kontrola skrutkového spoja

Na obrázku je znázornený proces kontroly skrutkového spoja. Systém v tomto prípade kontroluje prítomnosť skrutky a matice a ich správne usporiadanie. Na ľavom obrázku sú spoje vytvorené správne, na pravom je nesprávne namontovaná matica, čo spôsobí, že kontrola bude neúspešná.



Obr. 5: Kontrola skrutkového spoja [28]

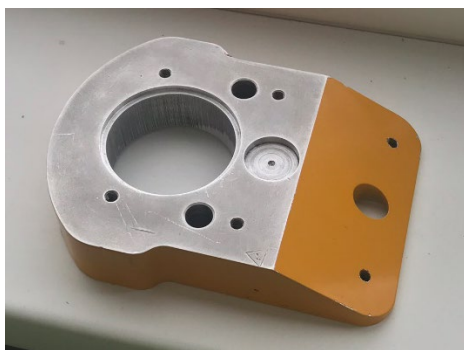
Ako vidieť z uvedených príkladov, možností použitia tejto technológie je naozaj mnoho. Postupným vývojom nových technológií sa presnosť a spoľahlivosť týchto systémov stále zlepšuje a môžeme predpokladať, že tento trend bude v budúcnosti ešte rásť.

1.3 Detekcia 3D objektov v priestore

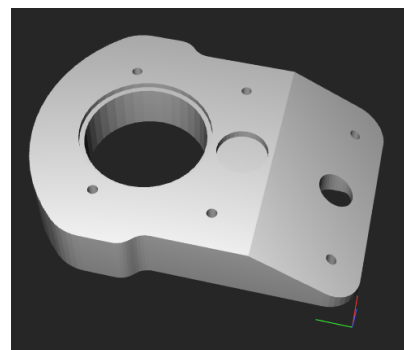
Rozpoznávanie 3D objektov pomocou jednej 2D kamery je značný problém kvôli strate informácií o hĺbke scény konverziou do 2D obrazu. Aby sme túto stratu informácie kompenzovali v praxi, používajú sa rôzne prístupy. Jednou z metód je použitie systému dvoch kamier, ktoré spolu vytvárajú tzv. stereo kameru. Na základe obrazu z dvoch kamier sa vhodným algoritmom dopočíta informácia o hĺbke v scéne. My sa však v tejto práci budeme zaoberať iba systémom s jednou kamerou. Pre určovanie 3D polohy objektu bude použitá programová knižnica s algoritmom poskytnutým spolupracujúcou firmou. V nasledujúcej podkapitole si popíšeme základný princíp tejto metódy detekcie a jej algoritmus.

1.3.1 Algoritmus detekcie

Poskytnutá programová knižnica používa na kompenzáciu stratenej informácie o hĺbke v scéne informáciu o hľadanom objekte. Ak chceme teda pomocou tohto programu 3D objekt zdetegovať, potrebujeme vlastniť jeho presný model v digitálnej podobe s odpovedajúcimi rozmermi. Algoritmus potom na základe 3D modelu objektu vytvorí šablóny, ktoré potom používa pri detekcii. Z vytvorených šablón a vstupného obrazu potom program určí polohu v priestore a natočenie objektu v súradnicovom systéme stola. Vstupný obraz kamery je najskôr prevedený na obraz hrán. V tomto obraze sa potom hľadajú vytvorené šablóny objektu a ukladá sa dosiahnutá zhoda. Šablóna sa vytvorí na základe hrán 3D modelu v zadanom rozsahu uhlov potrebných na detekciu. V tejto práci budeme ako testovací objekt používať nasledovný objekt poskytnutý spolupracujúcou firmou. Na obrázku vľavo je zobrazená reálna podoba a vpravo poskytnutý 3D model objektu.



a) fyzický objekt

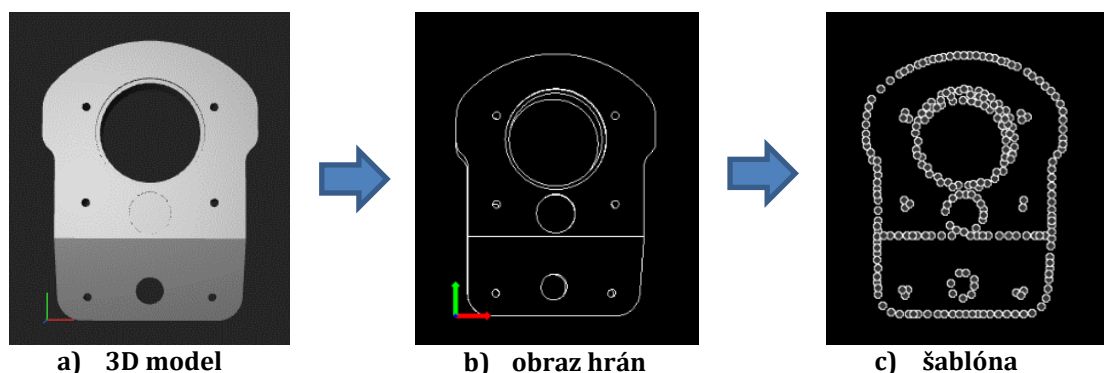


b) digitálny 3D model

Obr. 6: Testovací 3D objekt

Pre vytvorenie šablóny tohto objektu program v danom pohľade konvertuje 3D model na obraz hrán. Z obrazu hrán program následne vyberie stanovené množstvo bodov, ktoré budú danú šablónu popisovať. Tieto body sa potom použijú pri hľadaní

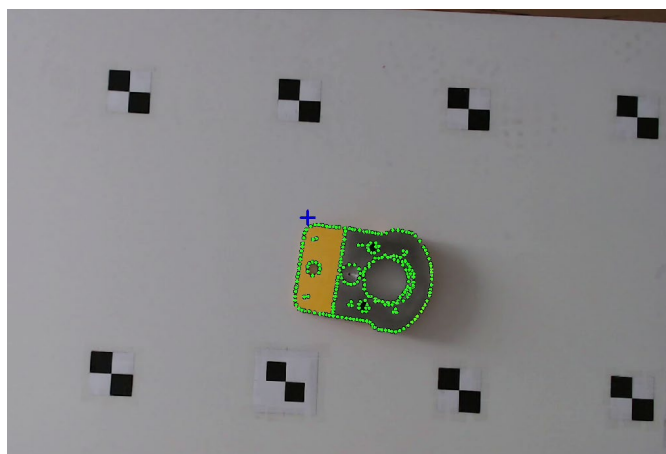
objektu v obraze kamery. Proces generovania šablón objektu je znázornený na nasledujúcom obrázku:



Obr. 7: Vytvorenie šablóny z 3D objektu

Algoritmus detekčného programu pri detekcii prechádza postupne všetky šablóny objektu a určuje zhodu s obrazom. Po spracovaní všetkých šablón program vráti najlepšiu percentuálnu zhodu. Ak je percentuálna zhoda väčšia ako stanovený prah, program vráti polohu objektu, v opačnom prípade je výstupom z programu informácia o tom, že hľadaný objekt sa nenašiel.

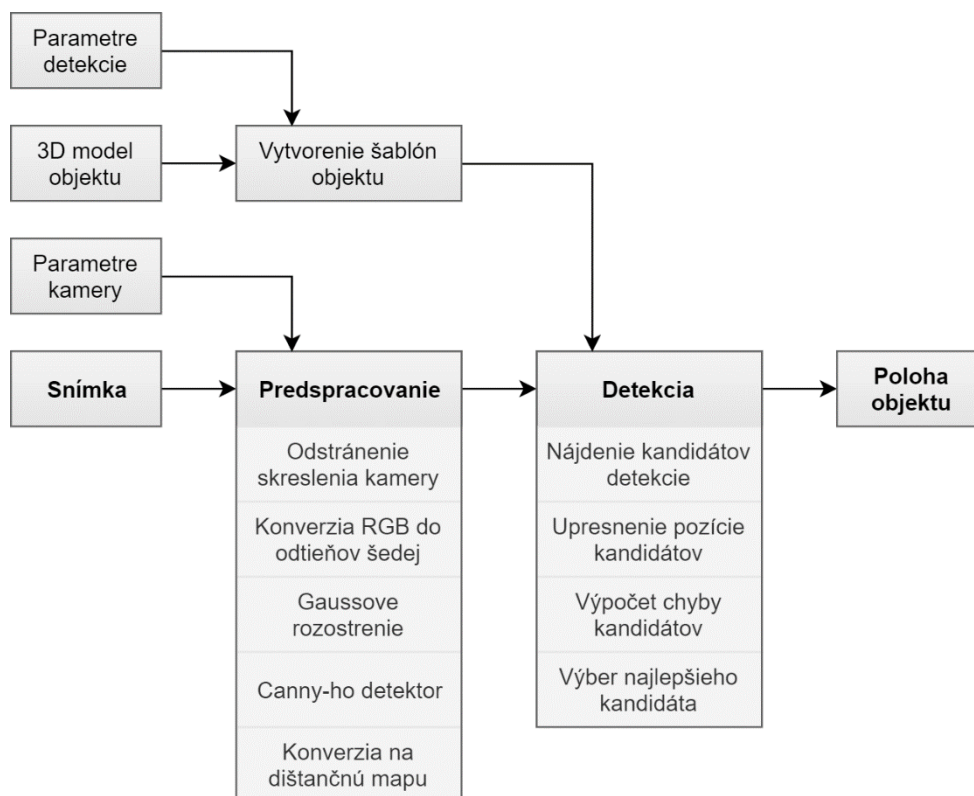
Na nasledujúcom obrázku je zobrazený príklad detekcie popísaného programu. Zelené body v obrázku označujú body šablóny s najväčšou zhodou. Modrý bod označuje počiatok súradnicového systému modelu objektu.



Obr. 8: Zobrazenie nájdeného objektu detektorom

Aby program správne určil polohu objektu, je nutné najskôr skalibrovať polohu kamery voči stolu. Toto je nutné vykonať vždy pri zmene polohy kamery tak, že na stôl umiestnime kalibračný plán, prípadne na stole môžu byť trvale umiestnené kalibračné značky. Tento spôsob bol použitý pri detekcii vo vyššie uvedenom príklade. Výstupná poloha, ktorú program vracia, je vždy odvodená od referenčného bodu použitého kalibračného plánu.

Celý algoritmus detekcie môžeme zakresliť do nasledovného blokového diagramu.

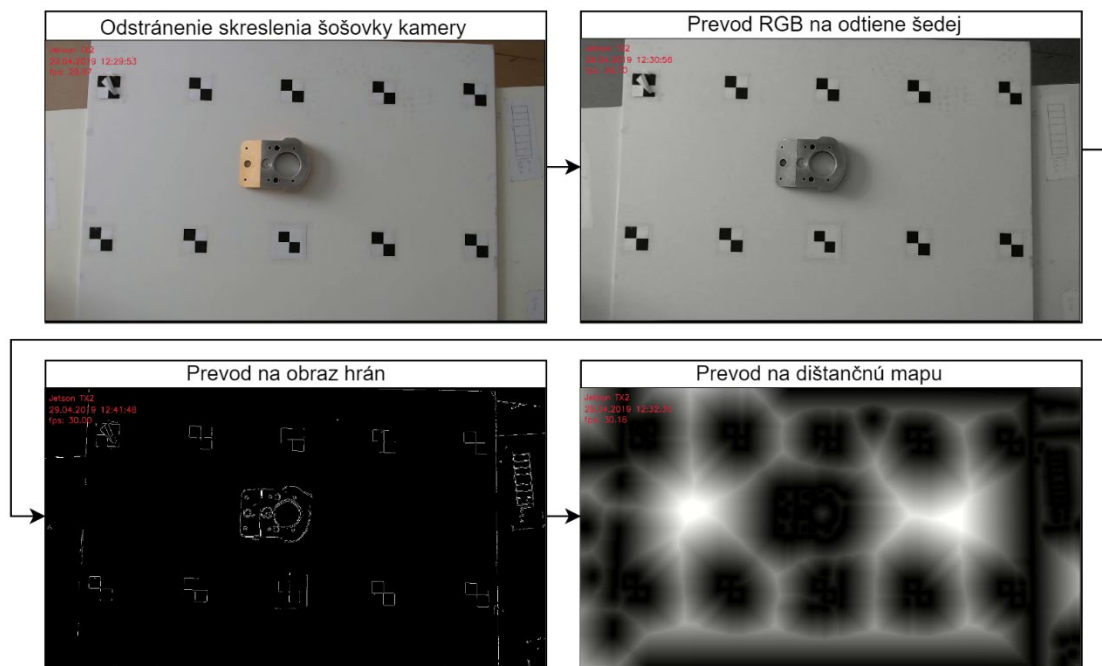


Obr. 9: Blokový diagram programu detekcie

V uvedenom blokovom diagrame môžeme vidieť priebeh detekcie spolu so všetkými potrebnými vstupmi. Aplikácia potrebuje pre správne fungovanie 4 vstupy:

1. Parametre detekcie,
2. 3D model objektu,
3. Parametre kamery,
4. Snímka z kamery.

Na základe zadaných parametrov detekcie a 3D modelu si program vygeneruje šablóny objektu. Parametre kamery musia obsahovať kalibračné parametre kamery, aby sa potlačili prípadné chyby šošovky. Po spustení detekcie program v slučke vykonáva kroky vyznačené hrubým písmom. Najskôr z kamery vyčítame aktuálnu snímku. Túto snímku na základe parametrov kamery upravíme a prebehne predspracovanie obrazu s krokmi uvedenými v blokovom diagrame. Výsledkom predspracovania je dištančná mapa. Jednotlivé kroky predspracovania na danej snímke je možné vidieť na nasledujúcom obrázku.



Obr. 10: Predspracovanie obrazu programom 3D detekcie

Nad takto upraveným obrazom spustíme algoritmus detekcie, ktorý pozostáva z niekoľkých krokov. Najskôr sa obraz prehladá a získa sa určitý počet kandidátov na detekciu. Algoritmus iteratívne prechádza celý obraz so stanoveným krokom a hľadá v ňom objekt pomocou vygenerovaných šablón. Tento krok je jeden z najviac výpočtovo náročných krokov detekcie. Výstupom z tohto kroku je zoradený zoznam kandidátov, ktorý dosiahli najvyššiu zhodu so šablónou objektu. V nasledujúcom kroku je spustená funkcia upresnenia polohy všetkých získaných kandidátov. V zadanom okolí každého kandidáta sa hľadá čo najlepšia zhoda so šablónou objektu. Týmto spôsobom sa nájde presná poloha objektu, ktorá nemusela byť v prvom kroku presne stanovená. Tento krok je výpočtovo najnáročnejší a v závislosti na počte nájdených kandidátov môže zaberat' aj niekoľkonásobne viac času, než predchádzajúci krok. Po upresnení polohy všetkých kandidátov sa vypočítajú chyby jednotlivých kandidátov od ideálneho stavu a stanoví sa percentuálna zhoda detekcie. Výstup z programu je kandidát s najväčšou zhodou.

Jednou z úloh tejto práce bude optimalizovať popísaný algoritmus detekcie a reťazec spracovania s použitím technológie CUDA, a tým dosiahnuť vyššiu rýchlosť stanovenia pozície objektu. Tento algoritmus bude potom implementovaný do navrhnutého systému na platforme Nvidia.

1.4 OpenCV

OpenCV je voľne šíriteľná knižnica pre počítačové videnie (angl. Computer Vision – CV) a strojové učenie. Knižnica bola vytvorená pre poskytnutie základnej infraštruktúry pre aplikácie s počítačovým videním a pre zrýchlenie vývoja strojového videnia v komerčných produktoch. OpenCV je poskytované pod licenciou BSD a je preto možné jej použitie a modifikácia aj v komerčnej oblasti.

Knižnica obsahuje viac než 2500 optimalizovaných algoritmov. Algoritmy môžu byť použité na nasledovné úlohy: detekcia a rozpoznanie tváre, identifikácia objektov, klasifikácia ľudských úkonov vo videách, sledovanie pohybu kamery, sledovanie pohybujúcich sa objektov, extrakcia 3D modelu objektov, spájanie obrázkov, hľadanie podobných obrázkov v databáze, sledovanie pohybu očí, rozpoznávanie scény a ďalšie.

OpenCV komunita má viac než 47 000 členov a odhadovaný počet stiahnutí knižnice približne 14 miliónov. Knižnicu používajú vo veľkej miere spoločnosti, vývojári a tiež vládne orgány.

OpenCV knižnica je dostupná v jazykoch C++, Python a podporu má tiež v matlabe. Knižnica je podporovaná pod systémami: Windows, Linux, Android a MacOS. [18]

1.4.1 OpenCV a CUDA

Knižnica OpenCV natívne podporuje technológiu CUDA a ponúka užívateľom rozhranie na využitie tejto technológie na zrýchlenie výpočtov pri práci s obrazom. Pre využitie tejto technológie pri spracovaní obrazu je nutné túto knižnicu zostaviť na vlastnom počítači, pretože zostavená verzia knižnice na oficiálnej stránke podporu CUDA neobsahuje. Postup ako knižnicu zostaviť s podporou CUDA a jej rozhranie pre prácu s dátami na grafickej karte si popíšeme detailnejšie pri samotnej realizácii. Technológiu CUDA si tiež podrobnejšie popíšeme v nasledujúcej kapitole.

V tejto práci budeme používať knižnicu OpenCV vo verzii 3.4, na ktorej je postavený aj popísaný program detekcie 3D objektov.

2 PLATFORMA NVIDIA

NVIDIA ponúka na trhu niekoľko výpočtových modulov pre vstavané zariadenia. Jedná sa o zariadenia, ktoré dosahujú výkonu grafických kariet a pri tom majú veľmi malé rozmery. Vďaka malým rozmerom a vysokému výkonu sú tieto zariadenia ideálna voľba pre inteligentné zariadenia ako roboti, drony, inteligentné kamery a prenosné zariadenia. Moduly majú vysokú paralelnú výpočtovú schopnosť, vďaka mobilnému čipu NVIDIA Tegra s vysokým počtom CUDA jadier. V súčasnosti ponúka NVIDIA nasledovné zariadenia ako platformu pre vstavané zariadenia:

- Jetson AGX Xavier,
- Jetson TX2 (TX2i),
- Jetson TX1,
- Jetson TK1.

NVIDIA ponúka modul TX2 aj vo verzii TX2i, čo je verzia určená pre priemysel. Táto verzia má zvýšený rozsah pracovných teplôt a je navrhnutá tak, aby vyhovela podmienkam v priemysle.

Prehľad parametrov dostupných zariadení na NVIDIA platforme možno vidieť v nasledujúcej tabuľke s vybranými parametrami, kompletná prehľadová tabuľka vlastností sa nachádza v prílohe Príloha A:

Developer Kit	Jetson AGX Xavier	Jetson TX2	Jetson TX1	Jetson TK1
GPU	NVIDIA Volta GPU, 512 Tensor Cores	NVIDIA Pascal™, 256 CUDA cores	NVIDIA Maxwell™, 256 CUDA cores	NVIDIA Kepler™, 192 CUDA cores
CPU	8-core ARM v8.2 64-bit	HMP Dual Denver 2 + Quad ARM® A57	Quad ARM® A57	NVIDIA 4-Plus-1 Quad ARM A15
Pamäť	16 GB 256-Bit LPDDR4x	8 GB 128 bit LPDDR4	4 GB 64 bit LPDDR4	2 GB 64-bit
Display	HDMI 2.0, eDP 1.2a, DP 1.4	2x DSI, 2x DP 1.2, HDMI 2.0, eDP 1.4	2x DSI, 1x eDP 1.4, DP 1.2, HDMI	HDMI
CSI	6x (16x) CSI-2	6x CSI-2	6x CSI-2	-
Úložisko	32GB eMMC 5.1, NVMe	32 GB eMMC, SDIO, SATA	16 GB eMMC, SDIO, SATA	16 GB 4.51 eMMC, SD/MMC, SATA
USB	USB 2.0, 2x Typ C - USB 3.1	USB 3.0, USB 2.0	USB 3.0, USB 2.0	USB 2.0 port, micro AB 1
Pripojenie	Gigabit Ethernet	Gigabit Ethernet	Gigabit Ethernet	Gigabit Ethernet
Rozmery	105 x 105 mm	50 x 87 mm	50 x 87 mm	127 x 127 x 26 mm
Výpočtová schopnosť	7.2 (podpora CUDA 10.0)	6.2 (podpora CUDA 10.0)	5.3 (podpora CUDA 10.0)	3.2 (podpora CUDA 10.0)
Cena modulu*	nedostupné	£405.36	£303.24	nedostupné
Cena kitu*	£1832.50	£441	£389.17	£140.15
Rok vydania	2018	2017	2015	2014

Tab. 1: Prehľad zariadení platformy Nvidia

* Ceny boli prebraté dňa 14.11.2018 zo stránky siliconhighwaydirect.co.uk. [5]

2.1 Nvidia CUDA

Jedným z pokynov práce je zamerať sa na multivláknové spracovanie obrazu. Na tento účel bude použitá technológia Nvidia CUDA, pomocou ktorej je možné zadané výpočtové úlohy vykonávať paralelne na procesore grafickej karty s veľkým počtom jadier. Týmto dosiahneme výrazné zvýšenie rýchlosti spracovania snímok.

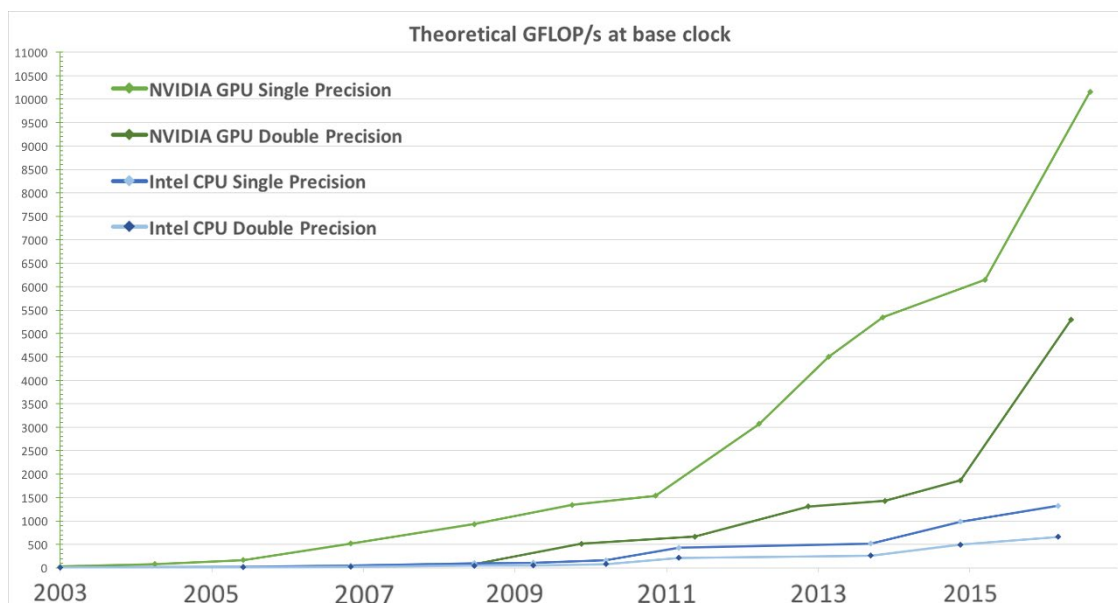
CUDA je paralelná výpočtová platforma a programovací model, ktorý umožňuje využiť výkon grafickej karty k bežným počítačovým výpočtom. Bola predstavená v roku 2006 a je možné ju použiť iba na grafických kartách od spoločnosti Nvidia. Použitím technológie CUDA môžeme využiť veľký výpočtový výkon GPU pre výpočty štandardne vykonávané procesorom. Týmto dosiahneme výrazný nárast výkonu pri paralelných výpočtoch. Nárast výkonu môže byť niekoľkonásobný, ale závisí hlavne od aplikácie a možnosti paralelizácie úloh.

Oblasť použitia tejto technológie je mnoho. Používa sa pri výpočtových úlohách, ktoré je možné rozdeliť a počítať paralelne. CUDA sa používa pre vedecké účely a vo vývoji, napríklad v týchto oblastiach: vizualizácia a spracovanie obrazu, neurónové siete, finančné modely, simulácie, numerické analýzy, bioinformatika a ďalšie oblasti, v ktorých sa dajú matematické výpočty paralelizovať.

Technológiu CUDA je možné využívať niekoľkými spôsobmi. Jedným zo spôsobov je použiť knižnicu, ktorá vnútorne používa CUDA kernel funkcie pre vykonávanie matematických operácií. Knižnica s podporou CUDA je v dnešnej dobe niekoľko. My budeme v práci využívať knižnicu OpenCV. Okrem týchto knižníc existuje mnoho ďalších, ktoré implementujú funkcie lineárnej algebry (cuBLAS, cuSPARSE, cuRAND,...), spracovanie signálov a obrazu (cuFFT, OpenCV, FFmpeg,...) a strojové učenie (cuDNN). Druhou možnosťou ako využiť technológiu CUDA je použitím rozšírenia CUDA v dostupných programovacích jazykoch. Podporovaných je niekoľko jazykov, medzi nimi je aj jazyk C++, v ktorom bude softvér tejto práce programovaný. CUDA pridáva do týchto jazykov niekoľko kľúčových slov, ktorými sa zabezpečí preklad napísaných funkcií pre spúšťanie na grafickej karte.

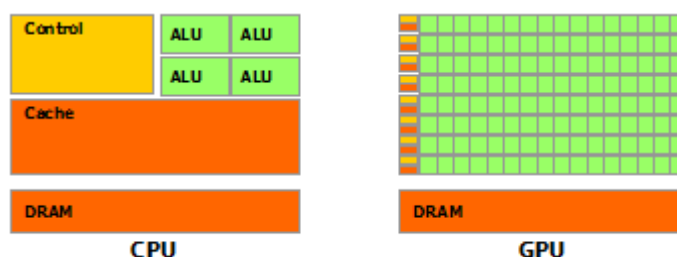
2.1.1 Programovací model CUDA

Požiadavky trhu pre 3D grafické aplikácie s veľkým rozlíšením podnietili vznik grafických kariet, ktoré sa vyvinuli do vysoko paralelných, multivláknových, niekoľkojadrových procesorov s obrovskou výpočtovou silou a veľkou šírkou pásma. Na nasledujúcom obrázku môžeme vidieť porovnanie výkonu procesorov Intel a Nvidia a ich vývin v spracovaní operácií s desiatinnou čiarkou. Na obrázku je vidieť, že výkon GPU v tejto oblasti za posledných niekoľko rokov začal výrazne prevyšovať výkon klasických CPU. [32]



Obr. 11: Porovnanie CPU a GPU pri operáciách s desatinnou čiarkou [32]

Dôvodom, prečo GPU niekoľko-násobne prevyšuje v tejto oblasti výkon CPU je, že GPU je špecializované na výpočtovo náročné paralelné úlohy. Za takýto vývin môže grafické renderovanie. GPU obsahuje viac tranzistorov určených na spracovanie dát, než na správu toku dát a pamäť cache. V porovnaní s CPU obsahuje oveľa viac aritmeticko-logických jednotiek (ALU), ako je schematicky naznačené na nasledujúcom obrázku:



Obr. 12: Porovnanie štruktúry CPU a GPU [32]

GPU je špeciálne navrhnuté na riešenie problémov, ktoré môžu byť interpretované ako dátovo paralelné výpočty. Rovnaký programový kód je spustený nad veľkým množstvom dát paralelne s vysokou aritmetickou intenzitou (označuje pomer medzi aritmetickými operáciami a operáciami s pamäťou). Pretože rovnaký program je spustený pre každý dátový element paralelne, nevzniká tu veľká požiadavka na riadenie toku dát ako v prípade procesora.

Dátovo paralelné spracovanie mapuje dátové elementy do paralelných vlákien. Mnoho aplikácií, ktoré spracováva veľké množstvo dát môžu použiť paralelný programový model na zrýchlenie výpočtových operácií. V prípade 3D renderovania sa jedná o veľké množstvo pixelov a vrcholov 3D objektov, ktoré sa

mapujú do paralelných vláken. Podobne sa toto dá aplikovať aj na spracovanie obrazu a médií pri rôznych operáciách: predspracovanie, zmena mierky, video kódovanie a dekódovanie, stereo videnie, rozpoznávanie vzorov, a iné... Pri všetkých týchto operáciách sa môžu mapovať pixely obrázkov do paralelných vláken. Okrem týchto sa používajú výhody paralelného spracovania aj vo finančných výpočtoch alebo v biológii.

CUDA bola predstavená ako paralelná výpočtová platforma pre všeobecné použitie a programový model, ktorý využíva výpočtový výkon Nvidia GPU na riešenie množstva komplexných výpočtových úloh efektívnejšie než CPU. S príchodom viacjadrových procesorov CPU a GPU sa tieto systémy stali paralelné a vznikol väčší nápor na programátorov vymyslieť spôsob ako užitočne využiť výkon všetkých jadier daného procesora. Najväčším problémom bolo navrhnúť softvér, ktorý transparentne prispôsobí svoje paralelné úlohy tak, aby využil narastajúci počet jadier procesora rovnako ako to robia 3D grafické aplikácie transparentne pri prispôbovaní sa GPU s veľkým množstvom jadier, ktoré sa v závislosti od typu môžu veľmi líšiť. Paralelný programovací model CUDA je navrhnutý tak, aby predchádzal týmto problémom.

Vo svojom základe CUDA poskytuje 3 kľúčové druhy abstrakcií – hierarchiu vláknových skupín, zdieľanú pamäť a bariérovú synchronizáciu, ktoré sú programátorovi poskytnuté ako minimálny set rozšírení používaného jazyka. Použitím týchto rozšírení je programátor vedený k rozdeleniu úloh na menšie „podúlohy“, ktoré môžu byť vyriešené nezávisle v paralelných blokoch a každá podúloha je rozdelená do menších častí, ktoré môžu byť kooperatívne vyriešené paralelne vláknami v rámci jedného bloku. Takáto dekompozícia zachováva expresivitu jazyka tým, že povoľuje vláknam spolupracovať pri riešení podúloh, a pritom poskytuje automatickú škálovateľnosť použitím blokov. Každý blok vláken môže byť spustený na ľubovoľnom multiprocessore v rámci GPU v akomkoľvek poradí, paralelne alebo sekvenčne, takže skompilovaný CUDA program môže byť spustený na ľubovoľnom počte multiprocetorov, ako je ilustrované aj na uvedenom obrázku vpravo.

Poznať koľko je na danej grafickej karte dostupných



Obr. 13: Rozdelenie CUDA programu na GPU

multiprocessorov nemusíme pri programovaní, ale stačí ho vedieť systému pri zavádzaní programu. Ten potom rozdelí jednotlivé bloky dostupným multiprocessorom. Rovnaký program teda môže na GPU s väčším množstvom multiprocessorov (SM) bežať rýchlejšie ako na GPU s menším počtom. Na uvedenom obrázku môžeme vidieť rozdelenie úloh CUDA programu pri rôznych grafických kartách.

Nasledujúci príklad znázorňuje jednoduchú aplikáciu rozšírenia CUDA pre zrýchlenie funkcie pôvodne vykonávanej na CPU.

Štandardný kód v C	C s rozšírením CUDA
<pre>void saxpy(int n, float a, float *x, float *y) { for (int i = 0; i < n, ++i) y[i] = a * x[i] + y[i]; } int N = 1 << 20; // Perform SAXPY on 1M elements saxpy(N, 2.0, x, y);</pre>	<pre><u>__global__</u> void saxpy(int n, float a, float *x, float *y) { int i = blockIdx.x * blockDim.x + threadIdx.x; if (i < n) y[i] = a * x[i] + y[i]; } int N = 1 << 20; cudaMemcpy(x, d_x, N, cudaMemcpyHostToDevice); cudaMemcpy(y, d_y, N, cudaMemcpyHostToDevice); // Perform SAXPY on 1M elements saxpy<<<4096, 256>>>(N, 2.0, x, y); cudaMemcpy(d_y, y, N, cudaMemcpyDeviceToHost);</pre>

Výpis 1: Príklad použitia CUDA v jazyku C

V zobrazenom príklade môžeme na ľavej strane vidieť kód napísaný v štandardnom jazyku C. Nachádza sa tu funkcia *saxpy*, ktorá robí jednoduché matematické operácie súčinu a súčtu s každým zadaným prvkom poľa. V tomto kóde sa výpočet vykonáva postupne pre každý prvok N-krát za sebou. V druhom prípade s využitím rozšírenia CUDA je funkcia napísaná pre beh na GPU a je vykonávaná paralelne na viacerých jadrách naraz. V porovnaní s funkciou pre C je v kóde pre GPU niekoľko príkazov naviac. Väčšina príkazov sa týka operácii s pamäťou, ktoré sú vyžadované pre beh. Premenné, ktoré budú využívané vo funkcii je nutné pred zavolaním skopírovať do pamäte GPU a po výpočte späť do operačnej pamäte. Ďalším hlavným rozdielom vo funkciách je direktíva „__global__“ pre prekladač, ktorý zaručí to aby táto funkcia bola preložená správnym kompilátorom CUDA. Koľkokrát bude funkcia spustená sa zadáva pri volaní funkcie do trojitých zátvoriek (<<< >>>). [14] [15] [16]

Použitie technológie CUDA má zmysel tam, kde zadanú úlohu môžeme rozdeliť na veľmi veľké množstvo menších úloh, ktoré sú na sebe nezávislé. Typickým príkladom použitia CUDA je napr. prahovanie obrázku, pretože hodnota každého pixelu je vypočítavaná nezávisle na ostatných a výpočet teda môžeme spustiť paralelne.

Aby výpočty, ktoré chceme presunúť na GPU skutočne zrýchlili beh našej aplikácie, treba zobrať do úvahy aj časovú náročnosť operácií s kopírovaním pamäte GPU a s réžiou GPU. Príklad, ktorý sme si uviedli, dosahuje vyššej rýchlosti spracovania na GPU až keď sa pohybujeme v ráde 10^6 výpočtov. Pri menšom počte výpočtov dosahuje lepšie výsledky CPU. Uvedený prah, pri ktorom sa oplatí v danej aplikácii použiť GPU je závislý od danej zostavy systému (CPU, GPU, RAM, ...).

2.1.2 Nvidia CUDA Toolkit

Nvidia CUDA Toolkit poskytuje vývojové prostredie pre vytváranie vysokovýkonných aplikácií s použitím grafickej karty pre zrýchlenie výpočtov paralelizáciou. S použitím nástrojov CUDA Toolkit je možné vyvíjať, optimalizovať aplikácie na GPU vo vstavaných systémoch, stolových počítačoch, dátových centrách, cloudových platformách a HPC superpočítačoch. CUDA Toolkit tiež obsahuje knižnice, odlaďovacie a optimalizačné nástroje, C/C++ kompilátor a knižnice pre publikovanie aplikácie. [13]

Najnovšia dostupná verzia nástrojov CUDA Toolkit je 10.0. Túto verziu budeme používať aj my v tejto práci.

2.2 NVIDIA JetPack

Nvidia JetPack je balík knižníc a vývojových nástrojov (SDK) pre platformu Jetson. Je to najrozsiahlejšie riešenie pre vytváranie aplikácií s umelou inteligenciou. Tento balík obsahuje všetko potrebné pre modul Jetson TX2, vrátane operačného systému a nástrojov pre host'ovský počítač, z ktorého je inštalácia vykonávaná. Pre inštaláciu tohto balíku do modulu Jetson TX2 potrebujeme host'ovský počítač s Linuxom. Jetson TX2 je potom k host'ovskému počítaču pripojený cez USB kábel. Po pripojení môžeme spustiť inštaláčného sprievodcu, ktorý nás prevedie celou inštaláciou. [17] V tejto práci budeme používať verziu JetPack 3.3 určenú pre Jetson TX2. Celý balík JetPack obsahuje nasledovné nástroje a knižnice:

- Jetson TX2 L4T operačný systém,
- Nástroje, dokumentácia, príklady použitia,
- CUDA,
- TensorRT (Knižnica pre strojové a hĺbkové učenie, zahŕňa klasifikáciu obrazov, segmentáciu a detekciu objektov neurónovými sieťami),
- cuDNN (CUDA Deep Neural Network – knižnica poskytuje základné prvky pre strojové učenie, obsahuje tiež podporu pre konvolúciu, aktivačné funkcie a tenzorové transformácie),
- VisionWorks/OpenCV (VisionWorks je vývojový balík pre počítačové videnie a spracovanie obrazu. Obsahuje základné funkcie počítačového videnia pre použitie s technológiou CUDA).

3 KOMUNIKÁCIA A PRENOS DÁT

V práci bude nutné riešiť prenos dát cez sieť (komunikácia medzi modulom a počítačom). Prenášať sa bude niekoľko typov dát, ktoré vyžadujú spoľahlivý prenos: riadiace príkazy, prenos súborov a ďalším typom dát bude obraz video kamery, ktorý nemusí byť spoľahlivý, ale vyžaduje čo najrýchlejšiu odozvu. Ďalej sa budeme v tejto kapitole venovať možnostiam zmenšenia potrebnej šírky pásma pre prenášanie obrazu pomocou video kompresie.

3.1 Komunikačné protokoly

Pre komunikáciu bude použité ethernetové rozhranie modulu a komunikácia bude používať IP protokol. Dve zariadenia v takto vytvorenej sieti môžu komunikovať pomocou rôznych protokolov. My si popíšeme 2 najčastejšie používané protokoly, ich oblasti použitia a ich výhody a nevýhody.

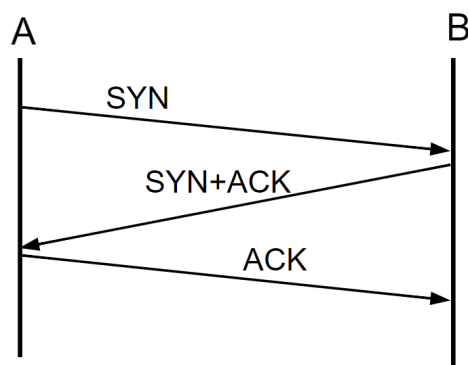
3.1.1 TCP protokol

TCP protokol (Transmission Control Protocol) je jedným z protokolov balíka internetových protokolov, ktoré tvoria jeho jadro. Protokol pracuje na transportnej vrstve ISO/OSI modelu. Protokol vznikol ako doplnok k protokolu IP a celý balík sa bežne označuje ako TCP/IP.

TCP poskytuje spoľahlivý a usporiadaný prenos s kontrolou chýb. Prenos prebieha po bajtoch medzi aplikáciami komunikujúcimi cez IP sieť. Hlavné aplikácie spoliehajúce sa na tento protokol sú napríklad: World Wide Web, email, FTP prenos súborov a iné. [19]

Hlavnou výhodou tohto protokolu je jeho spoľahlivý prenos. Pri prenose dát vždy obe strany vedia, či boli dáta úspešne prijaté alebo sa stratili pri komunikácii. Problémy, ktoré môžu nastať pri prenose (strata paketu, duplicita paketu, neusporiadaný príjem paketov), protokol deteguje a zabezpečí preposlanie dát alebo prijaté pakety zoradí do správneho poradia. Kvôli uvedeným vlastnostiam sa tento typ prenosu nehodí pre aplikácie vyžadujúce rýchlu odozvu prenosu ako napr. telefonovanie cez internet, videohovor. Pre takéto aplikácie sa používa protokol UDP.

Aby dve aplikácie mohli spolu komunikovať cez protokol TCP, musia najskôr nadviazať spojenie, ktoré prebieha podľa diagramu na obrázku vpravo:



Obr. 14: TCP Handshake [20]

Nadviazanie prebieha odoslaním paketu so signálom SYN, na ktorý druhá strana odpovie signálom ACK. Po nadviazaní spojenia, môžu aplikácie medzi sebou komunikovať, pričom každý paket, ktorý jedna strana odošle, je potvrdzovaný druhou stranou signálom ACK. Pre ukončenie komunikácie sa používa signál FIN. [20]

3.1.2 UDP protokol

UDP protokol (User Datagram Protocol) patrí rovnako ako TCP protokol do balíka internetových protokolov a tvorí jeho jadro. Protokol pracuje na transportnej vrstve ISO/OSI modelu. Pomocou UDP protokolu môžu aplikácie v IP sieti posielat' medzi sebou správy, ktoré sú označované ako datagramy. Pre odoslanie správy aplikácii v sieti nie je potrebné nadviazať komunikáciu ani vytvoriť kanál v porovnaní s TCP. UDP používa jednoduchý mechanizmus komunikácie bez nutnosti vytvárať spojenie. UDP implementuje kontrolu integrity dát a paket obsahuje číslo portu pre rozlíšenie zdrojovej a cieľovej aplikácie. Odoslané správy pri tomto protokole nie sú potvrdzované a protokol tak poskytuje nespoľahlivé spojenie. Nie je garantované, že správa bude doručená do cieľovej aplikácie a tiež nie je zabezpečené, že správy prídu v správnom poradí. Tento protokol nerieši ani problém duplicity prijatých správ.

Kvôli uvedeným vlastnostiam sa tento protokol používa pri komunikácii, ktorá nemusí byť spoľahlivá, ale ide nám predovšetkým o rýchlosť odozvy (telefonovanie cez internet). [21]

3.1.3 Knižnica Asio

Asio je multiplatformová knižnica napísaná v jazyku C++ pre prácu so sieťou, ktorá zabezpečuje rozhranie pre prácu s nízkoúrovňovými vstupmi/výstupmi. Asio tiež poskytuje základné stavebné bloky pre prácu so sieťou a procesmi/vláknami v C++. Asio je podporovaná v týchto systémoch: Linux, Windows, macOS, FreeBSD. Knižnica má implementovanú podporu pre komunikáciu na sieti IPv4 a IPv6 a transportné protokoly TCP, UDP a ICMP. [22]

Asio vytvoril Christopher M. Kohlhoff a je súčasťou balíka knižníc *Boost*. Asio sa dá na stránke autora stiahnuť aj ako samostatná knižnica. [22] Knižnica je poskytovaná pod licenciou *Boost Software License, Version 1.0* a vďaka nej môže byť použitá voľne, v ľubovoľnom projekte, s podmienkou zachovania hlavičiek dokumentov obsahujúcich informácie o autorovi a licencii.

My v tejto práci budeme knižnicu používať na vytvorenie komunikačných kanálov cez TCP a UDP na prenos dát a riadiacich príkazov medzi aplikáciami.

3.2 Prenos živého obrazu

Prenos živého obrazu medzi zariadeniami sa bežne označuje ako streamovanie. Jedno zo zariadení slúži ako server a druhé ako klient, prípadne môže byť pripojených viacero klientov na rovnaký server. Server dáta zapuzdrí a posiela cez ethernet v paketoch. Na posielanie sa používa viacero protokolov. Najčastejšie používaným protokolom je UDP a RTP. Pri týchto protokoloch sa nevytvára medzi zariadeniami overované spojenie a tak je prenos dát rýchlejší. Výsledkom je, že stream má menšie oneskorenie. Pri UDP sa ale môžu pakety strácať, čo nám v našom prípade nevadí, pretože stream bude slúžiť len ako náhľad obrazu kamery.

3.2.1 Kompresia dát

Prenášanie živého obrazu z kamery bez kompresie je vysoko dátovo náročné. Aby sa dátová náročnosť znížila, obraz sa komprimuje do rôznych video formátov. Na kompresiu videa bolo vytvorených niekoľko kodekov, ktoré dokážu výrazne znížiť dátovú náročnosť prenášaného videa. Každý kodek pri tom dosahuje iné výsledky a rôzne zaťažuje procesor. Nevýhodou kompresie je, že zaťažuje procesor servera, pretože výpočet je náročný. Náročnosť závisí od rozlíšenia a počtu snímok za sekundu prenášaného videa.

3.2.1.1 Kodek

Kodek je označenie používané pre softvér na kódovanie a dekódovanie obsahu video súborov. Na strane tvorca videa slúži na kódovanie videa do formátu kodeku. Na strane diváka kodek slúži na dekódovanie obsahu videa pre videoprehrávač.

V nasledujúcich odstavcoch si popíšeme tri často používané kodeky v súčasnosti a porovnáme ich vlastnosti z hľadiska kvality reprodukcie videa a výpočtovej náročnosti.

3.2.1.2 H.264/MPEG-4 AVC [23]

H.264 je blokovo orientovaný, video kompresný štandard, označovaný tiež ako AVC (Advanced Video Coding). V dnešnej dobe patrí k jedným z najrozšírenejších formátov používaných pri nahrávaní, kompresii a úschove video obsahu. Maximálne podporované rozlíšenie je 8K UHD. Účelom bolo vytvorenie štandardu schopného poskytovať dobrú kvalitu videa pri znížení dátovej náročnosti aspoň na polovicu oproti predchádzajúcim štandardom.

Štandard H.264 sa typicky používa pre stratovú kompresiu, avšak dajú sa vytvoriť regióny obrazu kódované bezstratovo alebo bezstratovo kódovať celé video. Tento štandard bol vyvinutý skupinou VCEG (ITU-T Video Coding Experts Group), v spolupráci so skupinou MPEG (ISO/IEC JTC1 Moving Picture Experts Group). H.264 je známy aj vďaka tomu, že sa stal jedným zo štandardov pre Blu-Ray

disky (všetky Blu-Ray prehrávače musia vedieť dekódovať tento štandard). Používajú ho aj niektoré webové streamovacie služby ako YouTube, Vimeo a tiež pri satelitnom, káblovom alebo terestriálnom vysielaní.

Pri použití H.264 sa znížila prenosová dátová náročnosť na polovicu v porovnaní s MPEG-2. Pri satelitnom vysielaní cez MPEG-2, ktorý sa používal pôvodne bola dátová náročnosť 3,5 Mbit/s. S použitím H.264 klesla dátová náročnosť na 1,5 MBit/s pri rovnakej kvalite.

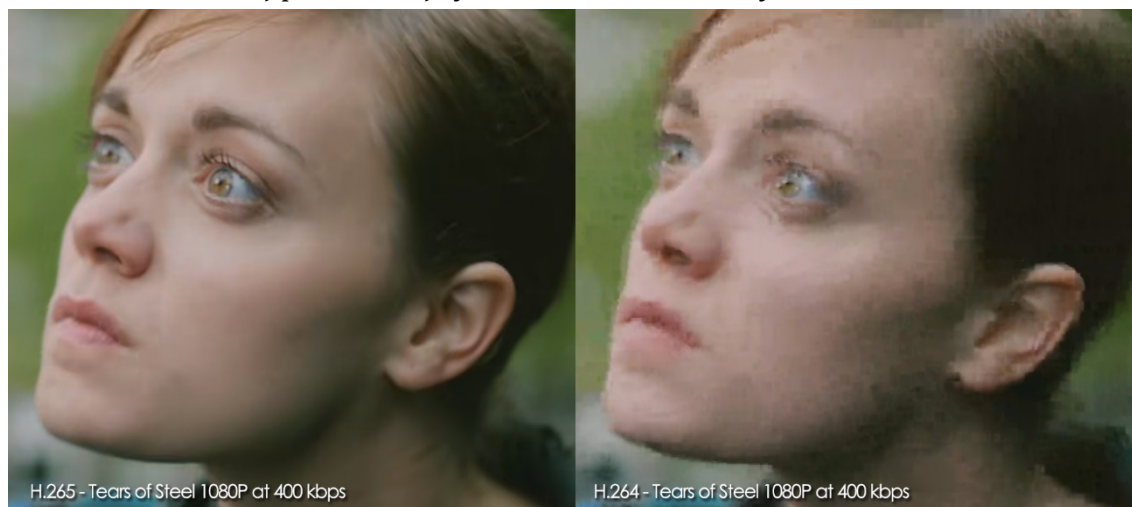
3.2.1.3 H.265/MPEG-H HEVC [24] [26]

Kodek H.265, označovaný tiež ako HEVC (High Efficiency Video Coding) je jeden z potenciálnych nástupcov štandardu H.264. Prvá verzia štandardu bola publikovaná v roku 2013. V porovnaní s H.264 ponúka štandard približne dvojnásobný kompresný pomer pri rovnakej kvalite videa alebo výrazne lepšiu kvalitu videa pri zachovaní rovnakej prenosovej rýchlosti. H.265 podporuje rovnako ako predchodca H.264 maximálne rozlíšenie 8K UHD.

Kodek pracuje podobne ako H.264. Porovnáva rôzne časti snímok videa a hľadá časti, ktoré sú redundantné, buď v rámci jednej snímky alebo série snímok. Tieto prebytočné oblasti sú potom nahradené krátkym popisom namiesto pôvodných pixelov. Hlavnou výhodou H.265 je približne dvojnásobná efektivita kompresie pri zachovaní rovnakej kvality videa. Znamená to, že H.265 videá prehrávané cez internet sa načítavajú podstatne rýchlejšie než H.264, pri zachovaní rovnakej kvality obrazu. Toto isté platí aj pre video súbory. Ak video kódované v H.264 bude mať 2GB, video v H.265 bude zaberáť len 1GB v rovnakej kvalite.

HEVC dosahuje dobré výsledky aj pri vyšších rozlíšeniach ako Full HD, napríklad pri rozlíšení 4K UHD sa dá dátový tok zredukovať až o 64%.

Na nasledujúcom obrázku môžeme vidieť porovnanie kvality videa pri zachovaní rovnakej prenosovej rýchlosti oboch uvedených kodekov:



Obr. 15: Porovnanie kodekov H.264 a H.265 (Full HD, 400kb/s) [25]

Popri všetkých výhodách, ktoré sme spomenuli má H.265 jednu nevýhodu. Najväčším problémom je výpočtová náročnosť kódovania. V priemere môže byť kódovanie H.265 videa až 10x náročnejšie pre procesor než H.264. Aj toto je jeden z dôvodov, prečo sa kodek H.264 stále vo veľkej miere používa. Dekódovanie videa nie je v porovnaní s kódovaním až tak náročné na procesor, ale oproti H.264 je potrebný približne dvojnásobný výpočtový výkon.

3.2.1.4 VP9

VP9 je video kodek s otvoreným zdrojovým kódom vyvinutý spoločnosťou Google ako nasledovník VP8. VP9 súperí v oblasti kódovania video obsahu hlavne s kodekom HEVC/H.265 od organizácie MPEG. Na začiatku bol VP9 použitý hlavne na Google video platforme Youtube, ale vďaka vzniku organizácii Alliance for Open Media a jej podpore na prebiehajúcom vývoji nasledovníka kodeku AV1, ktorej je Google súčasťou, viedol k rozšíreniu formátu VP9. V porovnaní s kodekom HEVC je VP9 rozšírené v takmer všetkých webových prehliadačoch a patrí mu väčšinová časť na trhu prehliadačov. VP9 dosahuje podobné kvality ako HEVC, pričom pri určitých parametroch môže byť výhodnejší jeden, pri inom nastavení druhý kodek. Aj kodek VP9 v porovnaní s H.264 požaduje niekoľkonásobne väčší výpočtový výkon na kompresiu videa ako bolo uvedené pri kodeku HEVC. [37]

3.2.2 FFmpeg

FFmpeg je multimediálna knižnica schopná kódovať, dekodovať, transkodovať, zmiešavať, rozdeľovať, streamovať a filtrovať takmer akékoľvek video. Knižnica je dostupná pod licenciou GNU GPL 3.0. Podporuje takmer všetky používané formáty, od najstarších až po tie najnovšie bez ohľadu na to, či boli navrhnuté štandardizujúcimi organizáciami, komunitami alebo spoločnosťami. Knižnica je tiež dobre prenositeľná a je možné ju skompilovať pod rôznymi platformami: Linux, MacOS, Microsoft Windows, BSD, Solaris a ďalšie. [27]

Táto knižnica ponúka enkodéry a dekodéry pri všetky popísané kodeky a využívať ich budeme v tejto práci pre dekodovanie obsahu. Kodeky nebudeme využívať priamo ale vnútorne sú zahrnuté v knižnici OpenCV.

3.2.3 GStreamer

GStreamer je softvérový framework pre vytváranie streamovacích aplikácií. Základný návrh pochádza z tzv. „video pipeline“, vytvoreného na Oregon Graduate Institute. Jedná sa o myšlienku vytvorenia potrubia z jednotlivých programových modulov, ktorými obraz prechádza. Niektoré myšlienky pri návrhu boli tiež prebraté z DirectShow. GStreamer umožňuje vytvoriť akúkoľvek streamovaciu

aplikáciu a je navrhnutý tak aby bola práca s ním čo najjednoduchšia. Primárne umožňuje spracovávať video a audio, ale môže spracovávať ľubovoľný dátový tok. Jedným z najčastejších využití knižnice je vytvorenie mediálnych prehrávačov. GStreamer obsahuje všetky potrebné komponenty na spracovanie širokého rozsahu vstupných formátov, napr. MP3, Ogg/Vorbis, MPEG-1/2, AVI, Quicktime a ďalšie.

Gstreamer poskytuje však toho omnoho viac, než potrebuje jednoduchý mediálny prehrávač. Celý framework GStreamer je postavený na moduloch-pluginoch, ktoré poskytujú rôzne funkcionality. Pluginy môžu byť spájané, čím sa vytvorí tzv. „pipeline“ (potrubie). Pipeline potom definuje dátový tok v danej aplikácii. Hlavnou kľúčovou funkciou knižnice GStreamer je poskytovať framework pre pluginy, spravovať dátové toky a zabezpečiť prepojitelnosť pluginov. Knižnica tiež poskytuje API pre vytváranie aplikácií použitím rôznych pluginov. [36]

Knižnica GStreamer bude použitá v tejto práci pri prenose videa po sieti. Bude vytvorená GStreamer pipeline, pomocou ktorej bude obraz prenášaný cez sieť do klientskej aplikácie.

4 NÁVRH SYSTÉMU

V nasledujúcej kapitole sa budeme venovať hardvérovému návrhu systému. Najskôr si uvedieme požiadavky kladené na navrhovaný systém, navrhujeme blokovú schému systému a potom vyberieme jednotlivé komponenty, ktoré podrobne popíšeme.

Aplikácia, ktorú sme si v teoretickom úvode popísali bola vyvinutá na klasickom stolnom počítači s OS Windows a v súčasnosti sa v tomto usporiadaní používa. Úlohou tejto práce bude celý systém presunúť do jedného šasi („krabice“) a vytvoriť tak kompaktné zariadenie schopné detegovať a rozpoznávať 3D objekty, ktoré bude možné nasadiť v rôznych aplikáciách. Celé zariadenie by teda malo dosahovať malých rozmerov.

4.1 Požiadavky na systém

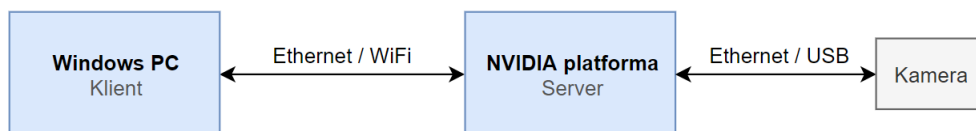
- Malé rozmery,
- Dostatočný výkon pre beh aplikácie detekcie 3D objektov,
- Dostatočne rýchle rozhranie pre pripojenie kamery,
- Schopnosť spracovať obraz s minimálnym rozlíšením Full HD pri 30fps,
- Rozhranie pre pripojenie k nadradenému systému (Ethernet, WiFi).

4.2 Bloková schéma systému

Systém bude zložený z troch blokov:

1. **Klientský počítač** – z tohto počítača budeme nastavovať a riadiť aplikáciu detekcie v module, po nastavení bude možné počítač odpojiť a systém bude pracovať s nastavenou konfiguráciou,
2. **Modul platformy Nvidia** – zariadenie, na ktorom bude spustený hlavný program detekcie, zároveň bude na ňom spustený TCP server, ktorý bude prijímať požiadavky z nadradenej aplikácie klientského počítača,
3. **Kamera** – vstup obrazu pre aplikáciu detekcie.

Bloková schéma systému, ktorý budeme navrhovať je nasledovná:



Obr. 16: Bloková schéma systému

4.3 Výber platformy

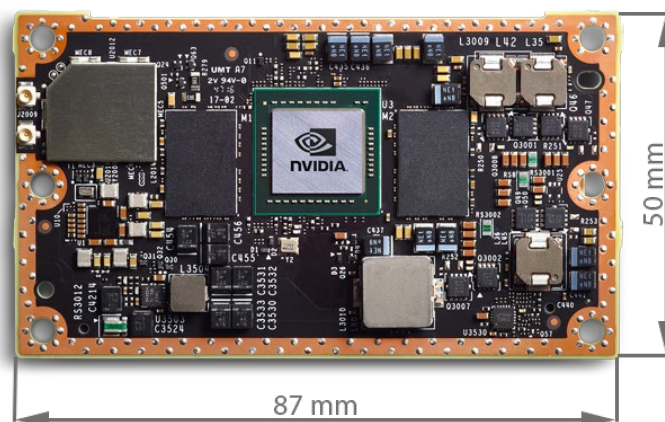
V dnešnej dobe je na trhu mnoho výrobcov, ktorí ponúkajú svoje zariadenia vhodné pre použitie vo vstavaných zariadeniach. My sa však zameriame len na zariadenia od spoločnosti Nvidia, kvôli kompatibilite programu s technológiou CUDA. Tieto zariadenia dosahujú veľmi malých rozmerov, a pri tom ponúkajú veľký výpočtový výkon.

Zo zariadení uvedených v teoretickom úvode v tabuľke *Tab. 1: Prehľad zariadení platformy Nvidia*, sme po konzultácii s vedúcim zamestnancom firmy vybrali modul TX2, vďaka jeho vyhovujúcim parametrom a cene. Celý vývojový kit spolu s modulom na testovanie sme objednali priamo z oficiálneho NVIDIA e-shopu za zníženú študentskú cenu na testovacie účely. Túto ponuku môžu využiť študenti a učitelia stredných a vysokých škôl a modul kúpiť v Českej republike za zníženú cenu 8 900Kč (štandardná cena je 16 565Kč).

4.3.1 NVIDIA Jetson TX2 (TX2i)

NVIDIA Jetson je jedno z najrýchlejších výkonovo-efektívnych vstavaných zariadení určených pre umelú inteligenciu (Artificial Intelligence) a aplikácie, ktoré vyžadujú malé rozmery pri zachovaní vysokého výkonu. Modul bol predstavený 7. marca 2017 na stretnutí v San Francisco, ako nasledovník modulu Jetson TX1 a stal sa vedúcou platformou na trhu. Jetson TX2 obsahuje integrované 256-jadrové Pascal GPU, 6-jadrový ARMv8 64-bit CPU komplex a 8GB LPDDR4 pamäte RAM s nízkou spotrebou energie. Komplex CPU kombinuje dva procesory: dvojjadrový *NVIDIA Denver 2* a štvorjadrový *ARM Cortex-A57*. Operačnému systému sa v tejto kombinácii procesor javí ako 6 jadrový. Typická spotreba modulu je len 7,5W.

Na nasledujúcom obrázku môžeme vidieť modul Jetson TX2 bez chladiča s uvedenými rozmermi. Ako vidíme, modul na svoje veľmi malé rozmery ponúka vysoký výpočtový výkon.



Obr. 17: Modul NVIDIA Jetson TX2 [4]

Jetson TX2 je postavený na 16nm *NVIDIA Tegra* architektúre (SoC) a dosahuje dvojnásobnú účinnosť v oblasti hĺbkového učenia než jeho predchodca Jetson TX1. Modul dosahuje výkon väčší ako CPU Intel Xeon v oblasti hĺbkového učenia. Tento skok v účinnosti mení možnosti migrácie cloudových riešení AI priamo na čip.

Jetson TX2 tiež podporuje video kódovanie a dekódovanie rôznych formátov. Podporuje tiež kodeky implementujúce H.265 kódovanie a dekódovanie pri rozlíšení 4K Ultra HD a 60 snímkach za sekundu. [6]

4.3.2 Vývojový kit pre Nvidia Jetson TX2

Nvidia ponúka pre rýchly vývoj aplikácií na module Jetson TX2 aj svoj vlastný kit. Na nosnej doske sú vyvedené takmer všetky rozhrania, ktorými modul TX2 disponuje. Modul je k nosnej doske pripojený pomocou 400-pinového konektora. V tejto konfigurácii je modul osadený chladičom, ktorý zabezpečí chladenie aj pri plnom zaťažení procesora. Na nosnej doske sú vyvedené nasledovné rozhrania modulu:

- 400-pinový port pre pripojenie modulu Jetson TX2,
- USB 3.0 Typ A,
- USB 2.0 Micro AB,
- HDMI,
- M.2 Key E,
- PCI-Express x4,
- Gigabit Ethernet,
- 802.11ac WLAN,
- Bluetooth,
- SATA,
- Slot pre SD kartu,
- UART, SPI, I2C, I2S, GPIO.



Obr. 18: Vývojový kit Nvidia Jetson TX2 [6]

Vývojová doska má rozmery 170,18mm x 170,18mm. Na uvedenom obrázku dosky sa modul Jetson TX2 nachádza pod chladičom. V tomto usporiadaní vidíme aké malé rozmery má modul v porovnaní s vývojovou doskou. Vývojový kit budeme v rámci práce používať pri vývoji a odladení softvéru pre naše zariadenie. Po odladení programového vybavenia bude nosná doska nahradená miniatúrnou doskou len s potrebnými konektormi.

4.3.3 Nosná doska

Keďže samotný modul TX2 neobsahuje konektory pre štandardné rozhrania (USB, Ethernet a iné), ale iba jeden prepojovací konektor, na ktorom sú všetky tieto rozhrania vyvedené, je nutné k modulu pripojiť nosnú dosku s potrebnými

rozhraniami. Nosné dosky pre modul nevyrába priamo Nvidia, ale jej partneri *ConnectTech* a *Auvidiea*. Tieto firmy ponúkajú niekoľko variant nosných dosiek a závisí už len od aplikácie, ktorú vývojár zvolí. Varianty nosných dosiek sa líšia hlavne vo veľkosti, v počte a type vyvedených rozhraní. V našej aplikácii bude kvôli požiadavke na malé rozmery použitá nasledovná nosná doska od spoločnosti *ConnectTech* obsahujúca všetky potrebné konektory.

4.3.3.1 Orbitty Carrier

Doska Orbitty Carrier je kompatibilná s modulmi Nvidia Jetson TX1, TX2 a TX2i a dosahuje rovnakých rozmerov ako tieto moduly. Základné parametre:

- Rozmery: 87 x 50 mm,
- Vyvedené rozhrania: HDMI, USB 3.0, USB 2.0, Gigabit Ethernet, microSD, 2x UART, 4x GPIO, I2C,
- Hmotnosť: 41g (nosná doska), 144g (nosná doska + modul s chladičom), 75g (nosná doska + modul bez chladiča),
- Cena: \$174.00 (v prepočte 3999,55 Kč, www.wdlsystems.com, 6.5.2019).



Obr. 19: Nosná doska Orbitty Carrier [39]

Na uvedenom obrázku je zobrazený modul Jetson TX2 bez chladiča spolu s nosnou doskou. Takéto usporiadanie bude použité aj vo finálnom zariadení. V zariadení bude modul spolu s nosnou doskou pripevnený na väčší chladič, kvôli odvodu tepla.

4.4 Výber kamery

Jedným z hlavných zariadení, ktoré budú k modulu pripojené, je kamera. Pre zabezpečenie spoľahlivého a dostatočne rýchleho prenosu z kamery, musíme vybrať vhodné rozhranie, ktorým kameru pripojíme.

4.4.1 Výber rozhrania

Vývojová doska s modulom Jetson TX2 ponúka nasledovné rozhrania na pripojenie kamery [3] :

1. USB 2.0,
2. USB 3.0,
3. Gigabit Ethernet,
4. CSI-2 (pripojenie až 6 kamier).

V nasledujúcich odsekoch si popíšeme jednotlivé rozhrania, spomenieme ich vlastnosti a porovnáme ich z hľadiska pripojenia kamery.

4.4.1.1 USB 2.0

USB 2.0 je dnes zatiaľ najrozšírenejšie používané rozhranie všeobecne v počítačovej technike. Väčšina dostupných webových kamier používa toto rozhranie. Hlavnými výhodami tohto rozhrania sú kompatibilita a dostatočná rýchlosť. Nevýhodou je, že tento štandard nepodporuje DMA, čo spomaľuje prenos údajov z kamery do pamäte počítača, a tak zvyšuje aj celkovú odozvu tohto rozhrania. Ďalšou nevýhodou tohto rozhrania je krátka vzdialenosť, na ktorú dokáže prenos fungovať. Maximálna vzdialenosť kamery a počítača pri prepojení cez USB 2.0 bez rozbočovača alebo aktívneho predlžovacieho kábla dosahuje len niekoľko metrov.

4.4.1.2 USB 3.0

Špecifikácia USB 3.0 je nasledovníkom USB 2.0. Teoretická priepustnosť rozhrania sa oproti USB 2.0 zvýšila 10-krát. Toto rozhranie používajú niektoré webové kamery, najmä však tie, ktoré vyžadujú väčšiu prenosovú rýchlosť a menšiu odozvu. Vďaka technológii DMA môže kamera zapisovať svoje dáta priamo do pamäte RAM počítača a nemusí čakať na operačný systém. Nevýhodou tohto rozhrania rovnako ako pri USB 2.0 je obmedzená dĺžka vedenia. Maximálna dĺžka dosahuje tiež len niekoľko metrov. Odporúčaná maximálna dĺžka sú 3 metre. Na dlhšiu vzdialenosť je nutné použiť aktívny predlžovací kábel. [2]

4.4.1.3 Gigabit Ethernet

Ethernet je rámcová technológia počítačových sietí pre lokálne siete LAN. Ethernet bol vyvinutý pre potreby prenosu dát medzi viacerými počítačmi. Prepojením počítačov tak vznikne lokálna sieť. Toto rozhranie prešlo dlhým vývojom, rýchlosť sa niekoľkonásobne zvýšila, pridali sa nové konektory – fyzická vrstva, ale koncepcia systému zostala rovnaká. Rozhranie Gigabit Ethernet je navrhnuté tak, aby dosahovalo rýchlosť prenosu 1 Gigabit za sekundu. Je pri tom ale nutné použiť vhodné káble. Pre dosiahnutie rýchlosti 1Gb/s je nutné použiť kábel minimálne typu *Cat-5e* s maximálnou dĺžkou 100m.

Veľké množstvo priemyselných kamier používa práve toto rozhranie. Výhodou je, že komunikácia funguje na veľkú vzdialenosť a môže sa tak s výhodou použiť aj v priemyselných halách, aj na dlhšie vzdialenosti. Kamera môže pri použití tohto rozhrania komunikovať s pripojeným PC pomocou klasických protokolov alebo použitím protokolu štandardu GigE Vision, ktorý niektoré spoločnosti implementujú do svojich kamier.

GigE Vision [9]

V roku 2006 bol predstavený štandard *GigE Vision*, ktorý poskytuje framework pre transport vysokorýchlostného videa a pridružených kontrolných dát cez Ethernet. Použitie tohto štandardu však nie je voľné a jeho distribúcia, vývoj a predaj

zariadení, ktoré implementujú tento štandard vyžadujú ročné predplatné. Tento štandard bol vytvorený skupinou 12 spoločností a neskôr sa výbor rozšíril na viac ako 50 členov. GigE Vision je založený na štandarde IP a pozostáva zo 4 elementov:

1. GigE Vision Control Protocol (GVCP) – Používa UDP protokol a definuje ako riadiť a nastaviť zariadenia. Špecifikuje komunikačné kanály pre streamovací mechanizmus transportu obrazu a konfiguračných dát medzi kamerami a počítačom.
2. GigE Vision Stream Protocol (GVSP) – Používa UDP protokol. Definuje dátové typy a spôsob, akým je obraz prenášaný cez GigE.
3. GigE Device Discovery Mechanism – Mechanizmus na získanie IP adries.
4. XML popisný súbor založený na schéme definovanej štandardom GenlCam, ktorý umožňuje prístup k ovládaniu kamery a obrazovému streamu.

Štandard GigE Vision využíva výhody rozhrania Gigabit Ethernet (vysoká prenosová rýchlosť, komunikácia na veľké vzdialenosti) a na prenos používa svoje vlastné protokoly optimalizované pre prenos videa, čím sa tento štandard radí medzi najlepších kandidátov na prenos videa. Ďalšou výhodou je možnosť kameru napájať priamo cez ethernetový kábel použitím technológie PoE (Power over Ethernet). Odpadne tak nutnosť ku kamere viesť dva káble, postačí iba ethernetový kábel.

4.4.1.4 Camera Serial Interface (CSI)

CSI je špecifikácia od organizácie MIPI Alliance, ktorá definuje rozhranie medzi kamerou a procesorom. Štandard prešiel vývojom a boli vytvorené 3 generácie štandardu, najnovšia je CSI-3. Najrozšírenejšia generácia je však CSI-2 a je použitá aj v module NVIDIA TX2 vo verzii 1.1, preto si ju bližšie popíšeme.

CSI-2 je možné implementovať na 2 rôznych fyzických vrstvách: MIPI C-PHY a MIPI D-PHY. V module NVIDIA TX2 je použitá ako fyzická vrstva MIPI D-PHY. Rýchlosť zbernice je v tejto verzii 2,5 Gb/s na vetvu, pričom zapojené sú 2 vetvy. Je tak možné prenášať z kamery obraz s rozlíšením 4K pri 20fps alebo 1080p pri 60fps bez kompresie (uvedené hodnoty boli dosiahnuté pri teste na stránke petermoran.org [10]).

Výhodou rozhrania CSI je, že je priamo navrhnuté a optimalizované na prenos videa z kamery do procesora a umožňuje tak dosiahnuť lepší výkon než iné rozhrania. Vďaka použitiu hardvérovej „vision pipeline“ odpadá nutnosť kopírovania obrazu do pamäte RAM a redukuje sa tak záťaž procesora. Nevýhodou tohto rozhrania je krátka dĺžka vedenia, ktorá závisí na prenosovej rýchlosti a pohybuje sa rádovo v desiatkach centimetrov.

4.4.1.5 Výber rozhrania pre pripojenie kamery

Po zohľadnení všetkých vlastností jednotlivých rozhraní modulu a po konzultácii s vedúcim zamestnancom firmy sme vybrali rozhranie Ethernet, vďaka svojej dobrej prenosovej rýchlosti a veľkej rozšírenosti tohto rozhrania. Väčšina priemyselných kamier je vybavených práve týmto rozhraním.

4.4.2 Výber kamery

Kamera je jedným z kľúčových komponentov zariadenia a jej výberu je potrebné venovať náležitú pozornosť, aby sme dosiahli požadované parametre. Úlohou kamery je generovať vstup pre reťazec spracovania obrazu. V našej aplikácii je požadované určiť čo najpresnejšie polohu objektu v obraze. Potrebujeme teda kameru s dostatočne vysokým rozlíšením, minimálne 1920 x 1080 (Full HD). Ďalším dôležitým parametrom kamery je rýchlosť zberu snímok kamery (framerate). Pre našu aplikáciu postačí kamera so štandardnou rýchlosťou 30 snímok za sekundu. Keďže objekty, ktoré budeme v obraze hľadať budú rôzne farebne odlíšené, budeme potrebovať kameru s farebným senzorom. Ďalším dôležitým parametrom pre nás je možnosť nastavenia správneho zaostrenia kamery. Keďže celé zariadenie bude pri použití pevne umiestnené, kamera nemusí nutne obsahovať automatické zaostrovanie, ale postačí manuálne. Pri výbere kamery budeme hľadať typ pre priemyselné použitie kvôli svojim definovaným vlastnostiam a kvalitnému spracovaniu. Po konzultácii s vedúcim zamestnancom firmy a zohľadnení spomenutých vlastností nám bola odporučená priemyselná kamera od firmy FLIR.

4.4.2.1 Kamera FLIR Blackfly S BFS-PGE-31S4C-C

Rada kamier *Blackfly S* kombinuje najnovší CMOS obrazový snímač s novou intuitívnou vývojovou platformou Spinnaker. Jedná sa o CMOS kameru s malými rozmermi, s rozhraním Ethernet a podporou štandardu GigE Vision. Kamera je vybavená mnohými funkciami a disponuje tiež funkciou automatického nastavenia expozície. Prehľad základných parametrov:

- Natívne rozlíšenie: 2048x1536,
- Max. snímková frekvencia: 35,
- Snímač: Sony IMX265, CMOS, 1/1.8",
- Globálna uzávierka,
- Rozhranie: GigE PoE,
- Štandard: GigE Vision v1.2,
- Spotreba: 3W,
- Rozmery: 29 x 29 x 30 mm,
- Cena: 495 USD (v prepočte cca. 11 244 Kč, www.ptgrey.com 2.1.2019).



Obr. 20: Kamera FLIR BFS-PGE-31S4C-C [30]

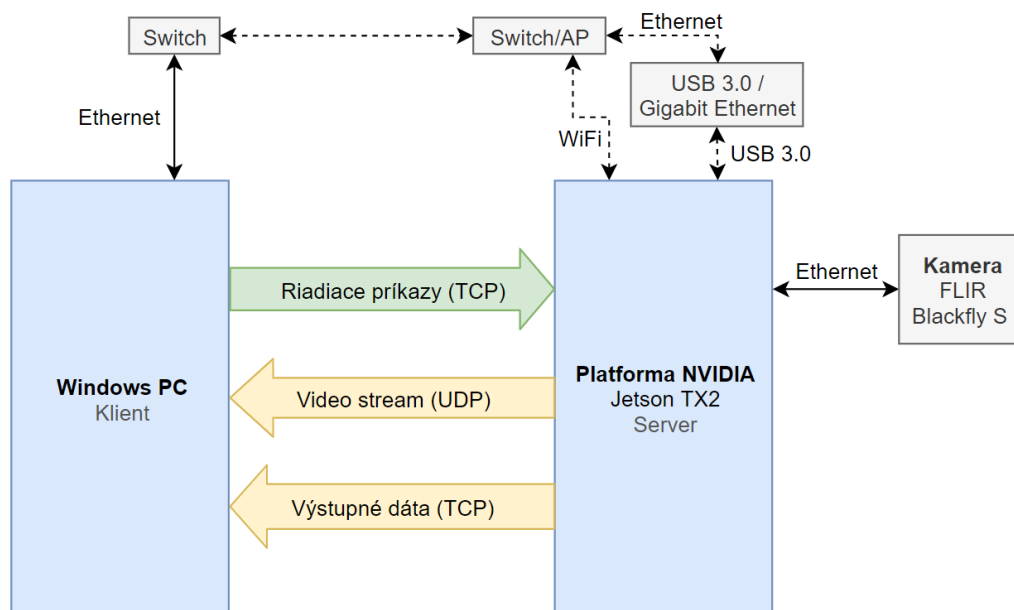
Kamera disponuje rozhraním Gigabit Ethernet s použitím štandardu GigE Vision. Pre dosiahnutie čo najmenšej odozvy pri komunikácii bude použitý samostatný ethernetový port na module. Kamera sa štandardne predáva bez objektívu, ktorý je nutné kúpiť samostatne. V našej aplikácii bude použitý objektív od firmy Tamron s označením LENS-80T4C. Tento objektív má fixnú ohniskovú vzdialenosť 8mm a poskytuje možnosť manuálneho ostrenia a nastavenie clony.

4.5 Komunikácia so zariadením

Zariadenie Jetson TX2 poskytuje niekoľko rozhraní vhodných na komunikáciu s inými zariadeniami (v našom prípade s nadradeným PC). Z rozhraní vyvedených na používanej vývojovej doske môžeme použiť na náš účel rozhranie USB 3.0 alebo rozhranie Gigabit Ethernet. Obe uvedené rozhrania ponúkajú dostatočnú dátovú priepustnosť pre realizáciu uvedených požiadaviek prenosu obrazu a príkazov. Z uvedených možností je pre nás najlepšou voľbou rozhranie Ethernet. Použitím tohto rozhrania na pripojenie zariadenia do siete získame možnosti pripojenia sa k zariadeniu aj na diaľku v rámci v rámci lokálnej siete napr. v rámci jedného podniku alebo po pripojení do internetu možnosť ovládať zariadenie z ľubovoľného miesta.

4.6 Výsledná bloková schéma

Po výbere všetkých potrebných komponentov sme zostavili blokovú schému s vybratými zariadeniami. V schéme sú naznačené hardvérové a aj virtuálne softvérové prepojenia.



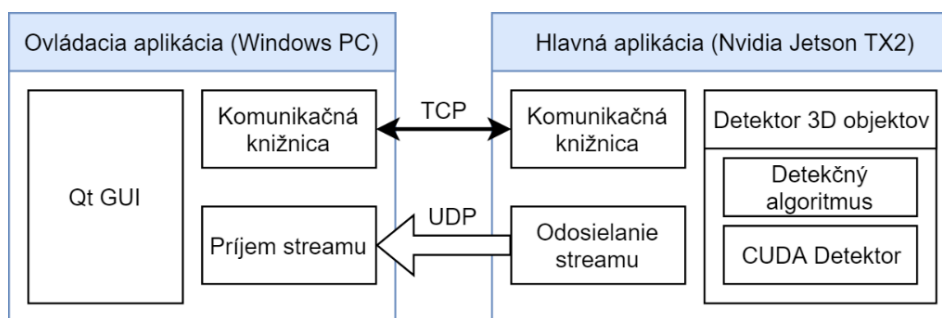
Obr. 21: Výsledná bloková schéma systému

Čiernou šípkou sú v schéme naznačené hardvérové prepojenia s označením použitého rozhrania. Čiarkovanou čiarou sú označené prepojenia, ktoré nemusia byť nutne použité. Farebné hrubé šípky (zelená a žltá) označujú softvérové virtuálne prepojenia (komunikačné kanály) medzi modulom a nadradeným PC.

5 NÁVRH SOFTVÉRU A IMPLEMENTÁCIA

Nasledujúca kapitola popisuje návrh a implementáciu softvérového vybavenia pre navrhnutý systém. Popísaná je vytvorená komunikačná knižnica slúžiaca na ovládanie modulu, ovládacia a hlavná aplikácia a úpravy programu 3D detekcie za účelom dosiahnutia multivláknového spracovania pomocou technológie CUDA.

V práci boli vytvorené dve samostatné aplikácie, pričom jedna slúži ako klient a druhá ako server. Klientská aplikácia je spustená na nadradenom počítači a ovláda sa ňou hlavná aplikácia, ktorá je spustená v module Jetson TX2. Klientská aplikácia poskytuje grafické rozhranie pre koncového užívateľa vytvoreného pomocou Qt SDK. Serverová aplikácia má implementovaný popísaný algoritmus 3D detekcie a ďalšie potrebné programové bloky zabezpečujúce riadiacu komunikáciu a prenos obrazu do klientskej aplikácie. Na nasledujúcom obrázku je zobrazená štruktúra programových blokov vytvoreného softvérového vybavenia.



Obr. 22: Štruktúra programových blokov

Pri vývoji softvérového balíku pre celú prácu sme kvôli bezpečnosti a pre zaistenie možnosti neskôr sa vrátiť k pôvodným častiam programu, všetky zdrojové kódy uchovávali v Git repozitári. Git je nástroj pre správu verzií softvéru, ktorý poskytuje vývojárom množstvo výhod. V tejto práci bol použitý hlavne kvôli ochrane dát. Celý repozitár je zálohovaný na „cloud“ a v prípade zlyhania disku môžeme celý repozitár stiahnuť cez web. Ďalšou vlastnosťou, ktorú sme pri vývoji použili je možnosť vrátiť sa k predchádzajúcim verziám programu. Git ponúka jednoduchý spôsob vrátenia sa k pôvodnej verzii programu, v prípade, že sme vykonali zmeny, ktoré boli nežiadúce.

Ako vývojové prostredie v tejto práci bolo použité prostredie Microsoft Visual Studio 2017, ktoré je pre študentov poskytované zadarmo. Prostredie poskytuje vývojárom užitočné nástroje pri vývoji softvéru. Jedným z najužitočnejších vlastností prostredia je technológia *IntelliSense*, ktorá vývojárom pomáha pri písaní kódu tým, že zobrazuje možnosti dopísania príkazov a zabráňuje tak preklepom v programe. Ďalšou užitočnou funkciou je balík odlaďovacích nástrojov. Tie nám

umožňujú aplikácie spustiť a za behu pozastavovať a sledovať požadované premenné a ich hodnoty.

5.1 Oživovanie vývojovej dosky Nvidia Jetson TX2

Vývojovú dosku Nvidia Jetson TX2, ktorú sme objednali bola hneď po vybalení funkčná a pripravená na použitie. Softvér, s ktorým bola dodaná, bol však už neaktuálny a preto sme spravili aktualizáciu na aktuálnu verziu 3.3. Pre aktualizáciu celého softvéru tejto vývojovej dosky ponúka Nvidia na svojej stránke softvér *JetPack Installer*. V tomto softvéri nakonfigurujeme všetky potrebné komponenty pre náš modul. Na výber máme zo všetkých komponentov popísaných v úvode v podkapitole Nvidia JetPack. *JetPack Installer* preinštaluje celý modul vrátane operačného systému a firmvéru. Aby bolo možné preinštalovať modul, je potrebné použiť host'ovský počítač s unixovým operačným systémom. Keďže počítač, ktorý sme mali k dispozícii používal systém Windows, inštaláciu sme vykonali pomocou virtuálneho systému. Na tento účel, bol použitý program VMWare Workstation 15 Player, ktorý spoločnosť VMWare poskytuje zadarmo pre študentov a nekomerčné použitie. Pomocou neho môžeme v operačnom systéme Windows spustiť virtuálny operačný systém. Ako operačný systém bol použitý Ubuntu vo verzii 16.04 LTS voľne dostupný na stránkach organizácie Canonical Ltd. Pre konfiguráciu a preinštalovanie modulu Jetson TX2 je potrebné, pripojiť ho k host'ovskému počítaču dodaným USB káblom, ktorý bude slúžiť na prenos dát. Pri inštalácii sme postupovali podľa návodu uvedenom na oficiálnych stránkach spoločnosti Nvidia. Odkaz na návod je uvedený v literatúre na konci práce [35]. Po inštalácii sme overili funkčnosť nainštalovaných nástrojov na priložených vzorových aplikáciách. Zariadenie bolo týmto pripravené a mohli sme prejsť k implementácii softvérového vybavenia.

5.2 Komunikačná knižnica

V nasledujúcej podkapitole sa budeme venovať návrhu komunikačného rozhrania, ktorým budeme serverovú aplikáciu ovládať. Keďže každá aplikácia bude spustená na inej platforme, je nutné použiť knižnicu, ktorá poskytuje platformovo nezávislé rozhranie pre prácu s TCP spojením. Po zohľadnení všetkých vlastností dostupných knižníc sme vybrali knižnicu *asio*, ktorú sme spomenuli aj v teoretickom úvode a plne vyhovuje našim požiadavkám.

Knižnica poskytuje vysokoúrovňové rozhranie napísané v štandarde jazyka C++11 pre vytváranie sieťových spojení. Hlavnou výhodou je, že poskytuje programátorovi abstraktné rozhranie zabezpečujúce na danej platforme správne spojenie so soketmi operačného systému. My v našej aplikácii budeme používať TCP

spojenie. Pre otvorenie TCP spojenia má knižnica definované vlastné abstraktné objekty. Najskôr si popíšeme vytvorenie TCP servera.

5.2.1 TCP Server

Pre vytvorenie ľubovoľného sieťového spojenia pomocou knižnice asio, musíme najskôr vytvoriť instanciu triedy *asio::io_context*, ktorý bude slúžiť ako základ pre všetky operácie vykonávané knižnicou. Ďalším krokom je vytvorenie objektu *tcp::acceptor*, ktorý zabezpečí otvorenie daného portu pre prichádzajúce spojenia. Knižnica poskytuje varianty metód objektov pre blokujúce aj neblokujúce volania. S výhodou použijeme asynchrónne volania a celé spracovanie požiadaviek sieťových spojení presunieme do vlastného vlákna. Najskôr však zavoláme metódu *async_accept()*, ktorou vytvoríme požiadavku na prijímanie spojení na danom porte. Ako parameter do tejto funkcie predáme objekt typu *tcp::socket*, ktorý bude reprezentovať vytvorené spojenie a funkciu, ktorú má program vykonať po úspešnom pripojení. Takto vytvorený program by zatiaľ nefungoval a ešte je potrebné vytvorený kontext knižnice spustiť. Na to slúži metóda *run()*, ktorú ako sme vyššie uviedli, spustíme vo vlastnom vlákne, pre zabezpečenie neblokujúceho volania.

Pre zabezpečenie pripojenia k serveru viacerým klientom naraz sa po úspešnom pripojení klienta na server, aktuálne spojenie presunie do zoznamu aktívnych spojení a program čaká asynchrónne na ďalšie spojenie. Takýmto spôsobom je zabezpečený teoreticky neobmedzený počet klientských spojení.

5.2.2 TCP Klient

Pre vytvorenie TCP klienta poskytuje knižnica asio obdobné objekty ako pre vytvorenie servera. Najskôr teda vytvoríme kontext, v ktorom budú vykonávané všetky operácie zabezpečujúce komunikáciu rovnako ako v prípade servera. Ďalším krokom je zavolanie funkcie *asio::connect()*, ktorej predáme instanciu objektu *tcp::socket*, ktorý bude reprezentovať spojenie a cieľový bod spojenia. Po návrate z funkcie skontrolujeme návratový kód a ak je všetko v poriadku pripojenie bolo úspešné. V takto otvorenom spojení môžeme teraz používať funkcie knižnice *read()* a *write()* a ich asynchrónne varianty pre posielanie dát.

5.2.3 Prenos správ

Po vytvorení TCP spojenia vyššie uvedeným postupom, môžeme medzi klientom a serverom posielat' textové alebo dátové reťazce. Reťazce, ktoré jedna strana odošle, nemusia byť druhou stranou prijaté naraz, ale môžu byť prijímané postupne. Je preto nutné si definovať vlastný formát pre posielanie správ s definovanou hlavičkou, ktorá zabezpečí správne prečítanie správy z dátového streamu. Na

posielanie krátkych textových správ bol navrhnutý formát paketu skladajúci sa z hlavičky a dát. Hlavička obsahuje informáciu o odosielanom type dát a na ďalšom riadku veľkosť správy v bajtoch. Pre názornosť si uvedieme príklad takejto správy a jej úpravu pre odoslanie cez TCP spojenie. Testovacia správa, „*test string message*“, sa pred odoslaním do dátového TCP streamu rozšíri o hlavičku, ktorá obsahuje typ posielanej správy a veľkosť správy v bajtoch. Hlavička je ukončená dvoma znakmi ukončenia riadku za sebou (v C++ zapísané ako “\n\n”). Celá správa bude mať tvar zobrazený v nasledujúcom výpise:

1	msg	// typ správy (konštantný údaj)
2	19	// veľkosť správy v bajtoch
3		// prázdny riadok
4	test string message	// dáta správy

Výpis 2: Formát posielanej správy cez TCP

Z takto odoslaných dát môžeme na druhej strane spojenia jednoducho vyčítať pôvodnú správu tým, že najskôr prečítame hlavičku správy. Na prečítanie hlavičky môžeme s výhodou použiť vstavanú funkciu knižnice *asio::async_read_until()*, ktorá do nami zadaného bufferu presunie všetky dáta až do oddelovacieho reťazca, predaného ako ďalší parameter. V našom prípade použijeme reťazec skladajúci sa z dvoch znakov nového riadku. Ďalším parametrom do tejto funkcie je funkcia, ktorú má program zavolať po úspešnom vyčítaní hlavičky, keďže sa jedná o asynchrónne volanie. V našom prípade tu predáme funkciu na vyčítanie dát prijatej správy. V tejto funkcii využijeme funkciu knižnice *asio::async_read()*, do ktorej zadáme presné číslo bajtov, ktoré chceme vyčítať na základe prijatej hlavičky. Po prijatí všetkých bajtov, funkcia zavolá nami definovanú obslužnú funkciu a pokračuje v čítaní ďalšej hlavičky.

5.2.4 Prenos súborov

Do hlavnej aplikácie v module Jetson TX2, bude potrebné posilať konfiguračné a prípadne ďalšie súbory týkajúce sa detekčného programu. Preto ďalšia funkcia, ktorá bola v komunikačnej knižnici implementovaná, je posielanie súborov. Posielanie súborov prebieha podobne ako posielanie správ popísané vyššie. Hlavička, použitá pri posielaní súborov, má nasledovný tvar:

1	file	// typ správy (konštantný údaj)
2	testfile.tmp	// názov súboru
3	250	// veľkosť súboru v bajtoch
4	781ca91c32d3972f71481f7991f85026	// kontrolný súčet MD5 súboru
5		// prázdny riadok

Výpis 3: Ukážka hlavičky pri posielaní súboru cez TCP

Za uvedenou hlavičkou potom nasledujú binárne dáta daného súboru. V porovnaní s hlavičkou pri posielaní správ sa v tejto hlavičke, okrem veľkosti súboru, odosiela

aj názov súboru a kontrolný súčet MD5. MD5 slúži na overenie, že pri prenose súboru nedošlo k jeho poškodeniu. Kontrolný súčet sa vždy pri odosielaní vypočíta a uloží sa do položky v hlavičke. Po prijatí súboru druhou stranou sa súbor uloží na dočasné miesto, kde sa spočíta nový kontrolný súčet MD5 a porovná sa s prijatou hodnotou. V prípade zhody je súbor posunutý programu na ďalšie spracovanie.

5.2.4.1 Kontrolný súčet MD5

Hashovací algoritmus MD5 je jeden zo skupiny algoritmov MD (Message Digest,) navrhnutých profesorom Ronaldom R. Rivestom z MIT (Massachusetts Institute of Technology). Algoritmus MD5 bol navrhnutý v roku 1991 po tom, čo bol algoritmus MD4 označený ako nedostatočne bezpečný, ako jeho bezpečná náhrada, aj keď v dnešnej dobe sa už tiež považuje za prelomený. Napriek tomu sa tento algoritmus často používa ako záruka, že prenášaný súbor dorazí nezmenený [33].

Zdrojové kódy pre výpočet MD5 boli prebraté z projektu *bzflag* a ich autorom je Frank Thilo. Zdrojové súbory sú dostupné na stránke projektu a používať ich je možné pri zachovaní uvedených podmienok. [34]

5.2.4.2 msgpack

Ďalšou funkciou, ktorá bola do knižnice implementovaná je posielanie ľubovoľných dátových štruktúr, teda aj nami definovaných užívateľských štruktúr. Na vytvorenie tejto funkcionality bola použitá knižnica msgpack.

Msgpack je knižnica, ktorá poskytuje efektívnu serializáciu ľubovoľných dát používaných v programe. Msgpack je podporovaná vo viac ako 50 programovacích jazykoch. My túto knižnicu budeme používať na prenos nami definovaných štruktúr. Msgpack požadovanú štruktúru rozloží na jednotlivé bajty, ktoré môžeme poslať pomocou popísaných TCP správ. Na strane príjemcu msgpack umožní znova zložiť prijaté dáta do pôvodnej štruktúry. Aby msgpack vedel nami definovanú štruktúru správne rozložiť a zložiť, musíme do štruktúry definovať makrom *MSGPACK_DEFINE()* premenné, ktoré sa majú prenášať. Príklad definície vlastnej štruktúry pre msgpack je uvedený v nasledovnom výpise:

1	<code>struct Person</code>	<code>// užívateľská štruktúra</code>
2	<code>{</code>	
3	<code> std::string name;</code>	<code>// meno</code>
4	<code> std::string lastName;</code>	<code>// priezvisko</code>
5	<code> int age;</code>	<code>// vek</code>
6	<code> MSGPACK_DEFINE(name, lastName, age);</code>	<code>// definujeme, ktoré elementy</code>
7	<code>};</code>	<code>// chceme zabaliť a ich poradie</code>
	<code>Person p = { "John", "Parker", 25 };</code>	<code>// testovacia štruktúra</code>

Výpis 4: Príklad užívateľsky definovanej štruktúry pre msgpack

Na takto definovanú štruktúru môžeme použiť štandardné funkcie knižnice. Pre zabalenie štruktúry použijeme funkciu *msgpack::pack()*, a podobne pre spätné

rozbalenie `msgpack::unpack()`. Štruktúra môže obsahovať len dátové typy, ktoré sú podporované. Podporované sú všetky vstavané typy jazyka a väčšina typov zo štandardnej knižnice. Ak chceme použiť nepodporovaný typ, musíme mu znova rovnakým spôsobom ako pri štruktúre vo výpise definovať makrom elementy, ktoré chceme zabaliť. Testovacia štruktúra `p` z výpisu má po zabalení knižnicou nasledovnú bajtovú reprezentáciu:

Index	0	1	2	3	4	5	6	7	8	9	10	11	12	13
Hodnota	147	164	74	111	104	110	166	80	97	114	107	101	114	25
Znak	-	-	'j'	'o'	'h'	'n'	-	'p'	'a'	'r'	'k'	'e'	'r'	-

Tab. 2: Dátová reprezentácia štruktúry po zabalení cez `msgpack`

V tabuľke sú uvedené hodnoty jednotlivých bajtov, ich indexy a ich odpovedajúci znak. Hrubým písmom sú vyznačené tie bajty, ktoré tvoria pôvodnú štruktúru. Vidíme, že `msgpack` pre zabalenie dát pridáva iba malé množstvo riadiacich bajtov (v našom prípade 3) a malé čísla, aj napriek tomu, že položka `age` je reprezentovaná typom `int`, ktorý zaberá štandardne 4 bajty, prenáša iba s použitím jedného bajtu. Tieto vlastnosti robia z knižnice veľmi efektívny nástroj pre prenos dátových štruktúr. Takto zabalené dáta v bajtovej podobe môžeme preniesť pomocou mechanizmu posielania správ definovanom vyššie.

V našej aplikácii sú funkcie tejto knižnice využité k prenosu dátových štruktúr potrebných pre detekčný program.

5.2.5 Aplikačné rozhranie knižnice

Pre prácu s komunikačnou knižnicou bolo vytvorené jednoduché rozhranie a definované 2 triedy: `KCom::Client` a `KCom::Server`. Priebeh nadviazania komunikácie sme si popísali v predchádzajúcej podkapitole pri popise knižnice asio. Teraz si popíšeme vytvorené rozhranie pre užívateľa knižnice.

Obe triedy obsahujú spoločné funkcie pre odosielanie dát `send()` a `sendFile()` a tiež funkciu `bind()`, ktorou zaregistrujeme obslužnú funkciu pre prijatý typ správy. Ak chceme teda vytvoriť rozhranie pre posielanie príkazov od klienta na server, na serveri zaregistrujeme funkciou `bind("cmd", &cmdHandler)` obsluhu prijatých správ typu „cmd“. Po prijatí príkazu zo serveru, program zavolá nami zaregistrovanú funkciu `cmdHandler()`. V nej potom na základe prijatého príkazu vykonáme potrebné operácie. Na strane klienta potom už iba zavoláme funkciu `send()` s parametrom „cmd“ a ako ďalší parameter dodáme samotný príkaz. Takto vytvorené rozhranie funguje obojstranne pre klienta aj server. Ak si klient zaregistruje príslušnú obslužnú funkciu, môže aj server posilať správy klientovi. Podobným spôsobom je riešený prenos užívateľských štruktúr. Pre každý typ štruktúry musíme klientovi alebo serveru zaregistrovať nový typ správy pre danú štruktúru a obslužnú funkciu. Na strane klienta potom opäť zavoláme funkciu `send()`

s prvým parametrom označujúcim štruktúru a ako druhý parameter predáme objekt štruktúry. Keďže *send()* je šablónová funkcia, prekladač nám pre každý typ použitej štruktúry vygeneruje pri preklade potrebnú funkciu.

Obdobným spôsobom je definované aj rozhranie pre posielanie a prijímanie súborov. Definované sú príkazy *sendFile()* a *bindFileHandler()*, pomocou ktorých môžeme odoslať alebo spracovať prijatý súbor. Pri posielaní súboru funkcií *sendFile()* predáme iba názov súboru a o načítanie a odoslanie sa postará funkcia. Pri prijatí súboru druhou stranou musíme definovať obslužnú funkciu s jedným argumentom typu *FileInfo*, ktorý obsahuje všetky informácie o súbore, vrátane dočasnej cesty k súboru. Prijatý súbor sa ukladá na dočasné miesto, z ktorého ho musíme pri spracovaní presunúť. V prípade, že prijatý súbor nevyužijeme, program ho pri vypnutí automaticky odstráni.

5.3 Prenos obrazu

Kvôli vizualizácii obrazu na klientskom počítači bolo nutné v práci vytvoriť komunikačný kanál na prenos obrazu. Pri návrhu sme najskôr zamýšľali nad použitím knižnice FFMpeg, ktorá bola popísaná v teoretickom úvode. Táto knižnica však nemá v OpenCV podporu pre vysielanie obrazu – streamovanie. OpenCV však pre streamovanie poskytuje modul GStreamer.

5.3.1 Odosielanie streamu

V teoretickom úvode sme si popísali knižnicu GStreamer a teraz si prakticky ukážeme spôsob vytvárania tzv. „pipeline“ reťazcov. Pipeline vytvoríme pospájaním jednotlivých modulov, pričom na začiatok pipeline pripojíme vstupný modul. Môže to byť video súbor, kamera, alebo iný obrazový vstup. Výstup celého reťazca pripojíme do nejakého zobrazovacieho modulu, prípadne do súboru. V našej aplikácii potrebujeme iba jednu pipeline pre streamovanie obrazu z OpenCV. Pre zabezpečenie prenosu videa medzi aplikáciami potrebujeme preniesť jednotlivé snímky z aplikácie bežiacej v module Jetson, kde sú reprezentované v podobe OpenCV matíc *cv::Mat*, do klientskej aplikácie v rovnakej podobe. Výhodou je, že knižnica OpenCV priamo podporuje GStreamer moduly, avšak v štandardnej verzii knižnice, ktorá je zostavená a dostupná na ich stránke, nie je tento modul zahrnutý a užívateľ si musí knižnicu zostaviť sám, aj s požadovaným modulom GStreamer. Zostavovanie knižnice OpenCV s neštandardnými modulmi si popíšeme neskôr, pri zostavovaní OpenCV s podporou CUDA.

Vďaka OpenCV podpore pre GStreamer môžeme teda v našom programe jednoducho vytvoriť objekt typu *cv::VideoWriter*, do ktorého budeme snímky zapisovať. Tento objekt bude prijímať snímky a posielat' ich do vytvorenej GStreamer pipeline.

5.3.1.1 GStreamer pipeline

Úlohou výstupnej pipeline bude snímky z našej aplikácie odoslať na UDP port sieťového rozhrania. Snímky z programu však nebudeme posilať priamo bez úprav, ale použijeme kompresiu dát niektorým z popísaných kodekov, čím znížime dátovú náročnosť. GStreamer pipeline sa v OpenCV zapisuje priamo ako textový reťazec, pričom jednotlivé moduly oddeľujeme znakom „!“. Prvým modulom v reťazci bude v našom prípade modul „appsrc“. Ten zabezpečí prevzatie snímky z objektu *cv::VideoWriter* a odoslanie do ďalšieho modulu reťazca. Keďže vstupné snímky, ktoré dostaneme sú vo formáte RGB, musíme ich najskôr konvertovať na formát, ktorý bude kodekom podporovaný. V teoretickom úvode sme si popísali 3 kodeky, z ktorých každý má svoje pozitíva aj negatíva. Všetky uvedené kodeky majú podporu Nvidia na module Jetson TX2 a môžeme tak pri kódovaní využiť hardvérovú akceleráciu. Na testovanie sme najskôr použili kodek H264, ktorý je v súčasnej dobe spomedzi nich najrozšírenejší.

Moduly pre GStreamer s podporou hardvérovej akcelerácie Nvidia sú súčasťou balíka modulov *gst-omx*, ktorý budeme používať. Tento balík obsahuje moduly pre kódovanie a dekódovanie formátov H264, H265, VP8, VP9, MPEG2 a MPEG4. Pre vytvorenie celého reťazca použijeme nasledujúcu konfiguráciu pipeline:

1	appsrc	// zdroj pipeline
2	! videoconvert	// Konverzia formátu
3	! video/x-raw,format=UYVY,framerate=30/1	RGB -> UYUY 30fps
4	! nvvidconv	// Konverzia rozlíšenia
5	! video/x-raw(memory:NVMM),width=1920,height=1080	na 1920x1080
6	! omxh264enc control-rate=2 bitrate=4000000	// kódovanie H264,
7	! video/x-h264,stream-format=byte-stream	4Mb/s
8	! h264parse config-interval=1	// H264 parser
9	! mpegtsmux	// zabalenie do MPEG
10	! rtpmp2tpay	kontajnera a vytvorenie
		RTP paketov
11	! udpsink host=127.0.0.1 port=5124	// odoslanie na UDP

Výpis 5: Výstupná GStreamer pipeline

Ako vidíme z výpisu, snímky do reťazca vstupujú modulom „appsrc“, následne prebieha konverzia obrazu na formát UYVY a nastaví sa počet snímok za sekundu na 30. Táto konverzia formátu je nutná pre pripojenie nasledujúceho modulu „nvvidconv“, ktorý obraz prenáša do grafickej pamäte a mení rozlíšenie vstupného obrazu na 1920 x 1080. Konverzia obrazu prebehne rýchlejšie, vďaka použitej hardvérovej akcelerácii Nvidia modulov *gst-omx*. Následne tento obraz vstupuje do modulu „omxh264enc“, ktorý obraz kóduje do formátu H264. Parametrom do tohto modulu je požadovaná prenosová rýchlosť prenosu, ktorá je nastavená ako konštantná s rýchlosťou 4Mb/s. Ďalšie pripojené moduly zabezpečia správne zabalenie obrazu do MPEG kontajnera, vytvorenie RTP paketov a nastavenie

intervalu posielania konfigurácie na 1 sekundu. Konfigurácia v sebe zahŕňa údaje o použítom formáte. Ak by sme tieto údaje neposielali periodicky, klient ktorý by sa pripojil po začatí prenosu, by nevedel dáta dekodovať.

Takto vytvorené pakety sú potom modulom „udpsink“ prenášané na požadované sieťové rozhranie.

5.3.2 Príjem streamu

Vďaka použitiu MPEG transportného streamu sú v odosielaných paketoch prenášané všetky informácie, ktoré príjemca potrebuje na dekodovanie a nemusíme ich explicitne pri prijímaní nastavovať. Na prijatie vytvoreného UDP streamu môžeme teraz použiť ľubovoľnú multimediálnu aplikáciu, napr. VLC a v ňom otvoriť UDP stream z danej adresy bez ďalších parametrov.

Pre príjem obrazu v programe budeme, rovnako ako pre vysielanie, používať triedy knižnice OpenCV. Na otvorenie vstupného obrazového zariadenia OpenCV poskytuje triedu *cv::VideoCapture*. Pripojenie k UDP streamu potom vykonáme zavolaním metódy *open("udp://127.0.0.1:5124")*, pričom použijeme IP adresu a port pripojeného modulu Jetson TX2. Z takto nastaveného objektu môžeme potom štandardným spôsobom vyčítavať snímky.

5.3.2.1 Výber kompresného formátu

Pri testovaní sme najskôr použili kodek H264. Pre zaistenie optimálneho prenosu obrazu boli niektoré parametre prenosu vyvedené do nastavení. V ovládacej aplikácii je možné nastaviť tieto parametre: rozlíšenie obrazu, snímkovú frekvenciu, bitovú rýchlosť a typ prenosu. Ďalej sme testovali kompresné formáty H265 a VP9. Ako bolo uvedené v teoretickom úvode, kompresia do týchto formátov je výpočtovo náročnejšia a ďalšou nevýhodou je, že modul reťazca pre gstreamer nepodporuje zabalenie týchto formátov do MPEG kontajnera a informácie pre dekodovanie formátu je nutné klientovi dodať externe, napríklad definovaním klientskej „pipeline“ pre gstreamer na klientskej strane. Aby sme zabezpečili, že prenášaný obraz bude môcť dekodovať príjemca aj bez knižnice gstreamer a bez nutnosti explicitne definovať parametre streamu, použijeme pre prenos iba formát H264.

5.3.3 Latencia prenosu

Pri konfigurácii zapojenia podľa schémy z Obr. 21, klientský počítač a modul Jetson TX2 prepojené cez ethernetový prepínač (switch), sme dosiahli pri použití kodeku H264 latenciu približne 640 ms. Táto hodnota predstavuje celkové oneskorenie zobrazeného obrazu v aplikácii voči reálnej scéne. Je v nej teda započítaný čas vyčítavania z kamery, čas spracovania, prenos po sieti, spracovanie klientskou aplikáciou a zobrazenie.

5.4 Hlavná aplikácia

V nasledujúcej podkapitole si popíšeme štruktúru hlavnej aplikácie, ktorá bude spustená na module Jetson TX2. Aplikácia bola vytvorená bez grafického rozhrania a celé ovládanie prebieha len pomocou príkazového riadku alebo vzdialene z klientskej aplikácie.

5.4.1 Preklad aplikácie v prostredí Linux

Na preklad a zostavenie zdrojových kódov hlavnej aplikácie v prostredí Linux potrebujeme vytvoriť súbor *Makefile*. Ten obsahuje informácie o zdrojových súboroch a potrebných knižniciach. Tento súbor môžeme vytvoriť a napísať sami, ale pri projektoch s viacerými zdrojovými kódmi a pri použití externých knižníc by tento proces mohol byť komplikovaný. Jednoduchší spôsob je použiť nástroj, ktorý tento súbor vytvorí za nás automaticky. Jedným z takýchto nástrojov je aj CMake.

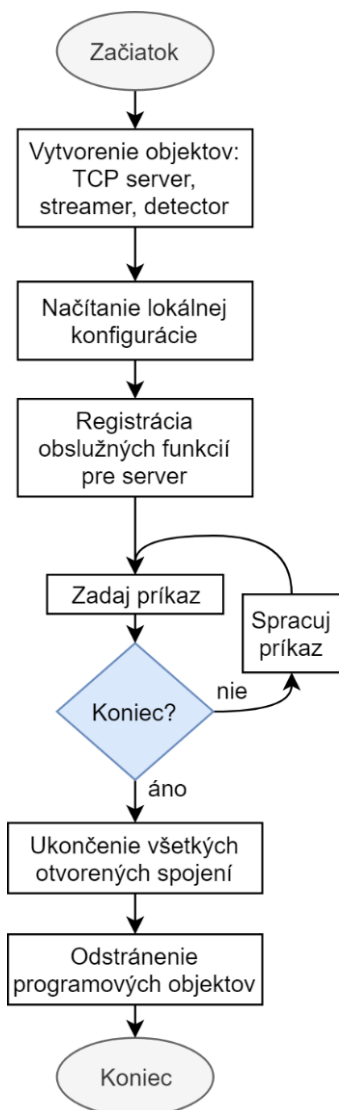
CMake je open-source multiplatformový set nástrojov, určených na zostavenie, testovanie a zabalenie softvéru. CMake sa používa na ovládanie procesu kompilácie softvéru použitím jednoduchej platformy a nezávislých konfiguračných súborov pre kompilátory a na vygenerovanie natívnych súborov pre zostavenie projektu v nami zvolenom vývojovom prostredí. Sada nástrojov CMake bola vytvorená spoločnosťou Kitware, ako odpoveď na potrebu výkonného multiplatformového zostavovacieho prostredia pre open-source projekty. [31]

Nástroj CMake je vhodné použiť hlavne pri multiplatformových projektoch, ktoré chceme zostaviť na rôznych operačných systémoch. CMake potom pri konfigurácii zostaví projektové súbory pre dané vývojové prostredie (napr. Microsoft Visual Studio v prípade OS Windows) alebo súbor *Makefile* pre unixové operačné systémy. Ďalšia funkcia, ktorú CMake poskytuje, je možnosť nakonfigurovať daný projekt podľa vlastných preferencií. Napr. môžeme navoliť požadované súčasti projektu a nepotrebné odstrániť, tým sa zníži čas kompilácie a výsledná veľkosť projektu.

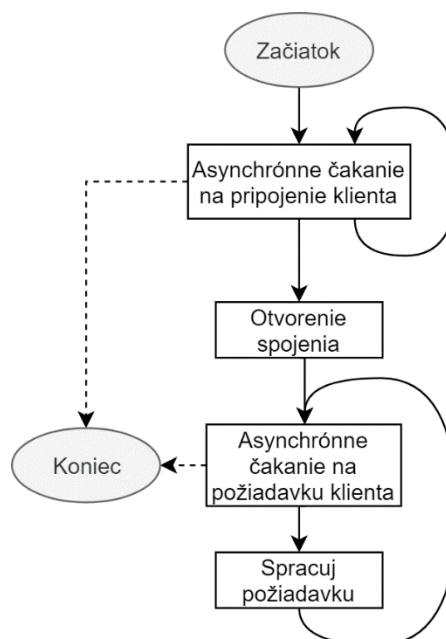
Celá konfigurácia projektu sa ukladá v súboroch s názvom *CMakeLists.txt*. Tento súbor je nutné vytvoriť pre každý projekt a nadefinovať súbory ku kompilácii, potrebné knižnice na zostavenie a doplniť ďalšie informácie zostavenia. Výhodou CMake je, že potrebné knižnice vie v rámci operačného systému dohľadať sám. CMake bol v rámci tejto práce použitý ako nástroj pre generovanie *Makefile* v module Jetson TX2.

5.4.2 Popis aplikácie

Celá aplikácia spustená na module Jetson TX2 pozostáva z niekoľkých programových blokov. Pre názornosť si uvedieme vývojový diagram celej aplikácie.



a) hlavné vlákno



b) vlákno pre TCP server

Obr. 23: Vývojový diagram hlavnej aplikácie

Na Obr. 23 je zakreslený vývojový diagram pre hlavné vlákno programu a vlákno pre TCP server. Hlavné vlákno po spustení vytvorí potrebné objekty: TCP server, streamer a detektor. Tieto objekty zabezpečujú potom všetky funkcie aplikácie: komunikáciu s klientom, stream obrazu kamery a detekčný algoritmus. Po inicializácii program čaká na zadanie užívateľských príkazov. Príkazmi sa od tejto chvíle riadi celý tok programu, až po ukončenie zadaním príkazu „exit“. Príkazmi sa riadi zapínanie a vypínanie streamu a detekcie, výber a pripojenie ku kamere, načítavanie konfiguračných súborov a ďalšie funkcie.

Po vytvorení objektu TCP servera sa spustí jeho vlákno a paralelne s hlavným vláknom sa čaká na pripojenie klienta. Po pripojení klienta zabezpečuje toto vlákno rovnaké funkcie spracovania príkazov ako hlavné vlákno.

Okrem popísaných vlákien obsahuje program ešte ďalšie 3 vlákna, pričom každé sa stará iba o jednu definovanú funkciu, ktorá beží v slučke. Kvôli ich jednoduchým vývojovým diagramom si ich nebudeme uvádzať, ale ich funkcie si popíšeme:

1. Vlákno – získavanie snímok z kamery (zabezpečuje, že pre slučku spracovania je vždy dostupná aktuálna snímka),
2. Vlákno – spracovanie obrazu (zabezpečuje spracovanie snímky algoritmom 3D detekcie a následné zaradenie snímky do fronty pre streamovanie),
3. Vlákno – vyčítanie snímky z fronty a odoslanie aktuálnej snímky do klientského počítača cez UDP protokol.

Vytvorený program obsahuje množstvo parametrov, ktoré môže užívateľ nastaviť. Aby sa parametre používané programom nestratili po vypnutí, je nutné ich ukladať. Na ukladanie parametrov bola vytvorená jednoduchá trieda, ktorá zabezpečuje ukladanie dát do súboru vo formáte XML. Využíva pritom voľne šíriteľnú knižnicu rapidxml [38]. Vďaka tejto triede sú nastavené parametre programu pri vypnutí automaticky uložené do súboru „config.xml“ a pri opätovnom zapnutí znova načítané.

5.4.3 Nvidia CUDA

V hlavnej aplikácii bola technológia CUDA použitá dvoma spôsobmi. Prvým spôsobom bolo použitie existujúcich CUDA funkcií v knižnici OpenCV. Štandardne zostavená knižnica OpenCV však natívne nepodporuje CUDA, preto si postup, ako do OpenCV pridať podporu pre CUDA popíšeme.

Druhým spôsobom využitia technológie CUDA bolo priame vytvorenie vlastných kernel funkcií, ktoré sa spúšťajú a vykonávajú priamo na grafickej karte paralelne. Postup vytvorenia vlastnej kernel funkcie si tiež popíšeme v nasledujúcich odsekoch.

5.4.3.1 CUDA v OpenCV

Knižnica OpenCV už samotná podporuje zrýchlenie výpočtov pomocou technológie CUDA, ale v štandardnom balíku, ktorý je dostupný na stránkach projektu, nie je CUDA zahrnutá. Na to aby sme mohli použiť CUDA v OpenCV, musíme knižnicu zostaviť sami aj s požadovaným modulom OpenCV. Modul Jetson TX2 však po nainštalovaní balíku JetPack podporu pre CUDA s OpenCV obsahuje, preto stačí vygenerovať OpenCV knižnicu pre počítač s OS Windows.

Pre zostavenie OpenCV s podporou CUDA najskôr zo stránky spoločnosti Nvidia stiahneme inštalačný balík CUDA a nainštalujeme ho. Následne zo stránky projektu OpenCV stiahneme zdrojové súbory knižnice a potom pomocou programu CMake popísaného vyššie nakonfigurujeme a vygenerujeme potrebné súbory pre kompiláciu knižnice. V tejto práci budeme využívať vývojové prostredie Microsoft

Visual Studio 2017, preto si pri konfigurácii zvolíme uvedený program a zvolíme zostavenie projektu pre 64-bitový systém. Pre vytvorenie projektu pre Microsoft Visual Studio použijeme program CMake s grafickým rozhraním. V programe najskôr nastavíme cestu k zdrojovým súborom a následne cestu k priečinku, do ktorého chceme súbory vygenerovať a klikneme na tlačidlo *Configure*. Po stlačení tlačidla sa nám zobrazí ponuka, kde vyberieme vývojové prostredie Microsoft Visual Studio 2017, zostava x64. Po dokončení konfigurácie sa nám v okne zobrazí veľké množstvo nastavitelných parametrov s červeným podfarbením. Červená označuje tie hodnoty, ktoré sa od poslednej konfigurácie zmenili alebo pribudli. Pre pridanie podpory CUDA, v okne zaškrtneme pole *WITH_CUDA* a znova prekonfigurujeme CMake projekt. Po dokončení konfigurácie sa nám zobrazia nové hodnoty s červeným podfarbením. Z týchto hodnôt zaškrtneme pole *CUDA_FAST_MATH* a znova projekt prekonfigurujeme. Teraz je CMake projekt pripravený a môžeme vygenerovať príslušný projekt pre Microsoft Visual Studio. O tom, že OpenCV bude kompilované s podporou CUDA vo verzii 10.1 nás informuje aj výpis programu CMake:

```

Custom HAL: NO
Protobuf: build (3.5.1)

NVIDIA CUDA: YES (ver 10.1, CUFFT CUBLAS FAST_MATH)
NVIDIA GPU arch: 30 35 37 50 52 60 61 70 75
NVIDIA PTX archs:

OpenCL: YES (NVD3D11)

```

Obr. 24: Výpis z programu CMake

Teraz môžeme program CMake zavrieť a otvoriť vygenerovaný projekt vo Visual Studiu. V ponuke Solution Explorer vyberieme projekt INSTALL a pravým tlačidlom spustíme zostavenie (Build). Knižnica OpenCV obsahuje veľké množstvo komponentov a v závislosti na vybraných komponentoch knižnice môže zostavenie celého projektu trvať aj niekoľko hodín. Na testovanom notebooku (DELL XPS 15 9560) zabralo zostavenie OpenCV približne 4 hodiny.

Pre použité knižnice OpenCV v našom programe musíme v projekte nastaviť cestu k priečinku „.../build/install/include“, ktorý obsahuje všetky potrebné hlavičkové súbory knižnice a nachádza sa v priečinku, ktorý sme zvolili pri konfigurácii. Ďalej je potrebné do projektu nalinkovať *lib* súbory knižnice OpenCV nachádzajúce sa v priečinku „.../build/install/x64/vc15/lib“. Týmto je konfigurácia dokončená a knižnicu OpenCV môžeme používať.

5.4.3.2 Implementácia OpenCV funkcií s podporu CUDA

OpenCV má definované abstraktné rozhranie na prácu s obrázkami pomocou matíc. Do matice nahráme obrázok, s ktorým pracujeme a na danú maticu aplikujeme požadované operácie. Klasická matica v OpenCV je definovaná triedou *cv::Mat* a jej obsah je uložený v operačnej pamäti RAM počítača. Všetky výsledky operácií

s maticami sa ukladajú do rovnakej pamäte. Ak však chceme použiť pre výpočet jadrá procesora grafickej karty, musíme matice preniesť do pamäte grafickej karty. Keďže kopírovanie pamäte z RAM do GPU RAM je relatívne pomalá operácia, presun dát na GPU, vykonanie výpočtu grafickým procesorom a následný presun späť do RAM, by výpočet nezrýchlil, ale naopak spomalil. Najvýhodnejšie je dáta matice uchovávať priamo v pamäti grafickej karty a výsledky vykonávaných operácií ukladať na rovnaké miesto. Na tento účel bol vytvorený v OpenCV nový typ matice, ktorej obsah je uchovávaný v pamäti grafickej karty a je definovaná triedou *cv::cuda::GpuMat*. Pre nahratie obsahu do takejto matice použijeme metódu *upload()* a pre spätné vyčítanie do obyčajnej matice metódu *download()*. Pre dosiahnutie čo najvyššieho zrýchlenia, je potrebné algoritmy navrhnuť tak, aby sa čo najmenej používali operácie kopírovania pamäte medzi pamäťou RAM a grafickou pamäťou. V našej aplikácii bolo zrýchlenie pomocou CUDA aplikované na predspracovanie snímok z kamery. Každá snímka z kamery prechádza nasledovnými operáciami:

1. Undistort – funkcia upraví vstupný obraz podľa parametrov kamery, čím sa potlačí skreslenie šošovky kamery,
2. Konverzia RGB obrazu do odtieňov šedej,
3. Gaussove rozostrenie – odstránime z obrazu šum,
4. Prevod na obraz hrán – aplikujeme Canny-ho detektor,
5. Konverzia na dištančnú mapu.

Takmer všetky uvedené funkcie majú aj svoje ekvivalenty pre výpočet na grafickej karte, okrem funkcie *undistort*. Túto funkciu bolo treba nahradiť dvoma funkciami, ktoré tak vykonajú vo výsledku rovnakú operáciu ako funkcia *undistort*. Prvým krokom je inicializácia mapy, ktorá bude slúžiť na prepočet pixelov v obrázku. To zabezpečí funkcia *cv::initUndistortRectifyMap()*. Táto funkcia nemá definovaný ekvivalent pre výpočet na grafickej karte, preto výpočet vykonáme na procesore a vygenerované mapy si uložíme a nahráme na grafickú kartu pre následnú aplikáciu. Ďalej použijeme funkciu *cuda::remap()*, ktorou príslušné pixely premapujeme. Táto funkcia je už podporovaná aj na grafickej karte, čo nám zabezpečí zrýchlenie výpočtu.

Ďalšou funkciou, ktorá sa nedala priamo nahradiť je prevod farby obrázku na odtiene šedej. Pre štandardné pomery farieb je možné použiť priamo vstavanú funkciu *cuda::cvtColor()* s parametrom *cv::COLOR_RGB2GRAY*. Aby sme ale zachovali možnosť definovať vlastné váhy pre konverziu na obrázok šedej, musíme použiť inú funkciu. OpenCV obsahuje funkciu *cv::transform()*, ktorá umožňuje vynásobiť každý pixel zadaným vektorom a výsledok skalárneho súčtu sa uloží ako aktuálna hodnota pixelu. Hodnota každého pixelu sa vypočíta podľa uvedenej funkcie:

$$dst(x, y) = m \cdot src(x, y) \quad (1)$$

m – vektor váh pre jednotlivé zložky farieb (štandardne používané váhy R: 0,299, G: 0,587, B: 0,114)

src – vstupný RGB obrázok

dst – výstupný obrázok v odtieňoch šedej

Ekvivalent tejto funkcie nie je pre použitie CUDA definovaný, preto na dosiahnutie požadovanej funkcie musíme použiť elementárne funkcie. Vstupný obrázok najskôr vynásobíme potrebnými váhami funkciou *cuda::multiply()*. Funkcia vynásobí každý kanál daného obrázka číslom zo zadaného vektora, rozdiel oproti funkcii *cv::transform()* je v tom, že funkcia nerobí skalárny súčin, ale násobí prvky elementárne, preto je výstupom obrázok s rovnakým počtom kanálov. Následne takto získaný obrázok rozdelíme na jednotlivé kanály RGB funkciou *cuda::split()* a kanály sčítame funkciami *cuda::add()*.

Ďalším krokom v predspracovaní je aplikovanie Gaussoveho rozostrenia na vstupný obraz. OpenCV obsahuje funkcia *cv::GaussianBlur()*, ktorá potrebnú funkciu vykoná. Pre aplikovanie Gaussoveho rozostrenia na obrázok uložený v grafickej pamäti používa OpenCV iný postup. Najskôr si musíme vytvoriť objekt triedy *cuda::Filter* použitím funkcie *cuda::createGaussianFilter()* s potrebnými parametrami. Funkcia alokuje na grafickej karte potrebné miesto pre filter a pre aplikáciu filtra na daný obrázok potom použijeme metódu triedy *apply()*. Účel vytvorenia objektu filtra je zabránenie zbytočného kopírovania medzi grafickou pamäťou a pamäťou RAM. Takto vytvorený objekt si potom uchováme a pri spracovaní ho už len aplikujeme.

Podobným spôsobom ako sme na obrázok aplikovali Gaussov filter, na obrázok aplikujeme Canny-ho detektor hrán. OpenCV pre vytvorenie obrazu hrán poskytuje funkciu *cv::Canny()*, ktorá nám vygeneruje obraz hrán. Na aplikovanie Canny-ho detektora pomocou CUDA najskôr vytvoríme objekt triedy *cuda::CannyEdgeDetector* a následne ho, podobne ako v prípade Gaussovho rozostrenia, aplikujeme na požadovaný obrázok.

Ďalším dôležitým krokom predspracovania obrazu je vytvorenie dištančnej mapy z obrazu hrán. OpenCV obsahuje aj na túto operáciu vstavanú funkciu *cv::distanceTransform()*. OpenCV neobsahuje ekvivalent funkcie, ktorá by bola urýchlená výpočtom na grafickej karte. Keďže sa ale jedná o relatívne rýchly výpočet, ponecháme tento výpočet na CPU.

5.4.3.3 Porovnanie rýchlosti predspracovania

Pôvodný reťazec predspracovania obrazu bolo nutné pre použitie s technológiou CUDA upraviť, pretože niektoré funkcie neboli dostupné. Úpravy, ktoré sme vykonali, zrýchlili aj samotné predspracovanie na CPU. Keďže modul Jetson TX2 podporuje rôzne módy procesora, meranie sme vykonali najskôr v štandardnom

móde, v ktorom sa nachádza procesor po štarte a v móde maximálneho výkonu. Čas predspracovania využitím CUDA v móde maximálneho výkonu sa podarilo skrátiť oproti pôvodnému času približne o 58,2% v móde max. výkonu. V porovnaní s upraveným CPU predspracovaním bolo CUDA predspracovanie rýchlejšie približne o 26,3%. Porovnanie rýchlostí predspracovania je možné vidieť v nasledujúcej tabuľke.

Typ predspracovania	Čas predspracovania [ms]	
	Mód 3	Mód 0
Pôvodný reťazec CPU	240 ms	134 ms
Upravený reťazec CPU	130 ms	76 ms
Reťazec CUDA	90 ms	56 ms

Tab. 3: Porovnanie rýchlostí predspracovania

Hodnoty uvedené v tabuľke boli merané na module Jetson TX2 v rôznych módoch procesora. Jednotlivé módy si popíšeme v nasledujúcom odseku.

5.4.3.4 Módy procesora v module Jetson TX2

Jetson TX2 podporuje rôzne módy nastavenia režimu procesora, kvôli pokrytiu širokého rozsahu výkonových a výpočtových požiadaviek danej aplikácie. Na module Jetson si môžeme zvoliť, ktoré jadrá procesora chceme nechať bežať, a ktoré chceme nechať vypnuté. Môžeme tak výrazne ovplyvniť spotrebu celého zariadenia. Modul ponúka v základe 5 módov procesora, ale skúsený užívateľ si môže nadefinovať aj svoje vlastné. V nasledujúcej tabuľke je zobrazený zoznam základných režimov.

Mód	Názov	Denver 2		ARM A57		GPU
		Jadrá	Frekvencia	Jadrá	Frekvencia	Frekvencia
0	Max-N	2	2.0 GHz	4	2.0 GHz	1.30 Ghz
1	Max-Q	0		4	1.2 Ghz	0.85 Ghz
2	Max-P Core-All	2	1.4 GHz	4	1.4 GHz	1.12 Ghz
3 *	Max-P ARM	0		4	2.0 GHz	1.12 Ghz
4	Max-P Denver	1	2.0 GHz	1	2.0 GHz	1.12 Ghz

Tab. 4: Zoznam módov procesora modulu Jetson TX2 [40]

Modul Jetson TX2 automaticky pri zapnutí nastaví mód č. 3 – Max-P ARM, ktorý je v tabuľke označený znakom *. V tomto móde je procesor Denver 2 vypnutý a beží iba ARM procesor. Pre dosiahnutie maximálneho výkonu zariadenia je nutné prepnúť na mód 0 – Max-N. V tomto móde sú povolené všetky jadrá oboch procesorov a sú pretaktované na vyššiu frekvenciu. Rovnako aj GPU je pretaktovaná na vyššiu frekvenciu. Pri zapnutí tohto módu je potrebné počítať so zvýšenou

spotrebou modulu a zabezpečiť dostatočné chladenie. Pre dosiahnutie minimálnej spotreby modulu slúži mód 1 – Max-Q. V tomto móde sú povolené len jadrá procesoru ARM a sú oproti štandardným hodnotám podtaktované, rovnako aj GPU.

Na nastavovanie módov v module slúži príkaz *nvpmodel -m [číslo módu]*. Ak chceme teda nastaviť maximálny výkon zariadenia, spustíme príkaz *nvpmodel -m 0*.

5.4.3.5 CUDA kernel

OpenCV funkcie popísané v predchádzajúcej podkapitole používajú štandardné C/C++ rozhranie a užívateľ funkcií nezasahuje priamo do kódu, ktorý je vykonávaný na grafickej karte, ale zrýchlenie cez CUDA používa vďaka knižnici OpenCV. V nasledujúcej podkapitole sa teraz budeme priamo venovať návrhu funkcií, ktoré budú spúšťané na grafickej karte a popíšeme si návrh CUDA kernelov.

CUDA kernel je funkcia, ktorá sa spúšťa paralelne priamo na grafickej karte. Funkcia je definovaná direktívou „`__global__`“, ktorá označuje, že funkcia nebude prekladaná štandardným prekladačom C/C++, ale použije sa prekladač *nvcc* (Nvidia Cuda Compiler). Zdrojové súbory, do ktorých môžeme písať kernel funkcie musia mať príponu *.cu*. Týmto dáme prekladaču vedieť, že sa jedná o zdrojový kód s CUDA funkciami a má byť kompilovaný špecializovaným prekladačom.

Pre zrýchlenie detekčného programu potrebujeme časti programu, ktoré sú výpočtovo najnáročnejšie prepísať s použitím CUDA. V popísanom programe na 3D detekciu sú dve funkcie, ktoré sú výpočtovo náročné a kód, ktorý vykonávajú je možné spustiť paralelne, pretože jednotlivé výsledky nie sú na sebe navzájom závislé.

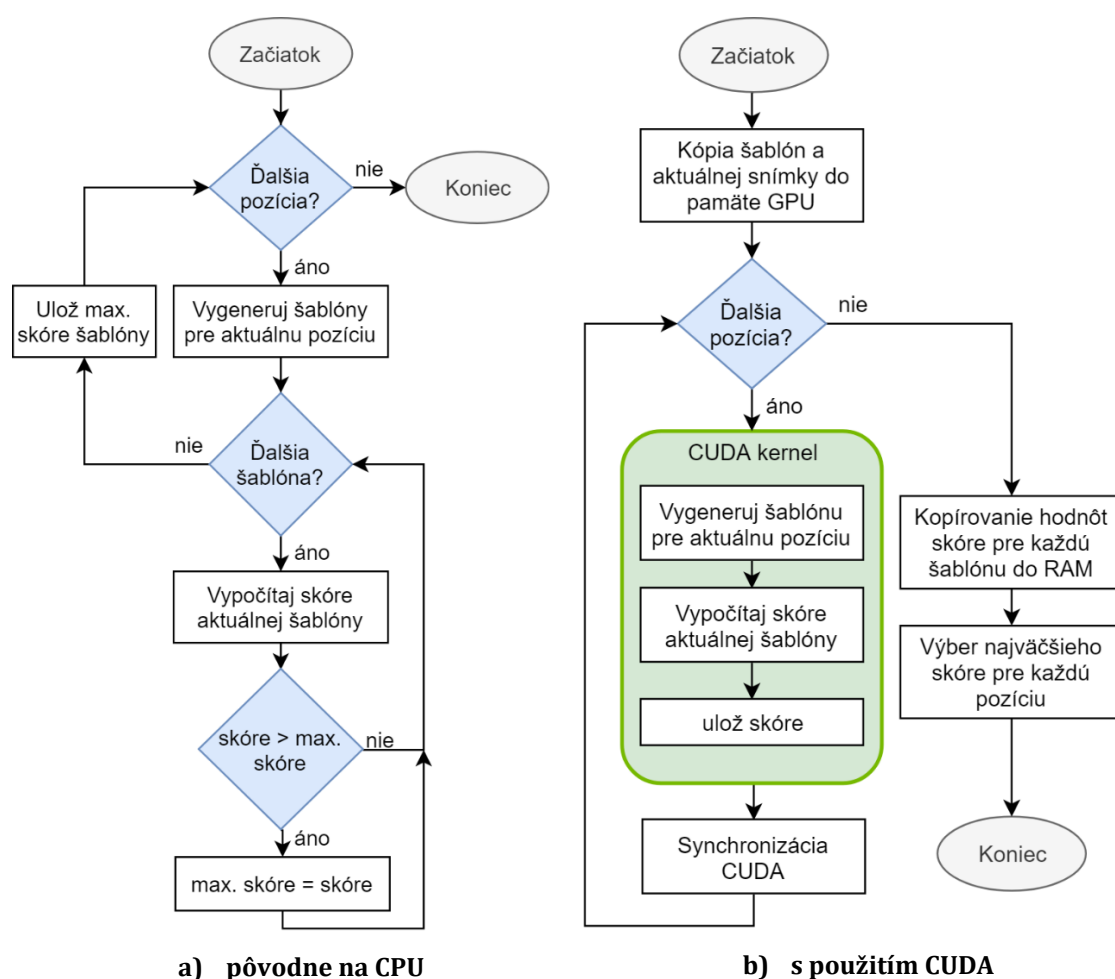
Prvou funkciou, ktorú budeme optimalizovať je funkcia *findCandidate()*. Ako bolo popísané v teoretickom úvode, táto funkcia hľadá vo vstupnom obraze vhodných kandidátov pre detekciu objektu. Funkcia prehľadáva celý obraz so stanoveným krokom. Funkcia pre každú pozíciu v obraze, ktorých môže byť rádovo niekoľko tisíc, porovnáva všetky šablóny objektu a ukladá dosiahnuté skóre. Šablón objektu môže byť v závislosti na zvolených parametroch detekcie tiež niekoľko tisíc. Toto nám dáva spolu po vynásobení rádovo $10^6 - 10^8$ rovnakých, navzájom nezávislých výpočtových operácií. Ako bolo popísané vyššie, použité CUDA má zmysel pri veľmi vysokej paralelizácii úloh, čo daná úloha spĺňa.

5.4.3.6 Aktuálna implementácia hľadania kandidátov

Aktuálna implementácia na CPU používa program, ktorý môžeme popísať pomocou vývojového diagramu zobrazeného na nasledujúcej strane na Obr. 25 a). V diagrame vidíme dve slučky, ktorých kód sa opakuje. Jedna slučka zabezpečuje prechod všetkými pozíciami a druhá prechod všetkými šablónami v danej pozícii. Pre každú šablónu sa vypočíta zhoda so snímkom v danej pozícii a vypočíta sa skóre. Výpočet skóre závisí na zvolenej metóde, ktorú je možné definovať v parametroch detekcie.

Po prechode všetkých šablón sa šablóna s najlepším skóre uloží ako najlepší kandidát pre danú pozíciu a pokračuje sa v slučke vygenerovaním šablón pre ďalšiu pozíciu. Funkcia končí po prechode všetkých pozícií a jej výstupom je zoznam najlepších kandidátov pre každú pozíciu.

V uvedenom pôvodnom programe nie je vnútorná slučka nezávislá, kvôli použitiu výberu maximálneho skóre po každej iterácii. Upravením programu tak, aby sa všetky skóre ukladali nezávisle a presunutím vyhodnotenia až po prejdení všetkých šablón dostávame slučku, ktorej jednotlivé iterácie nie sú na sebe vzájomne závislé a operácie vykonávané vo vnútri slučky môžeme spustiť paralelne. Môžeme teda prejsť k vytvoreniu CUDA kernelu, ktorý bude vykonávať výpočet aktuálneho skóre pre danú šablónu.



Obr. 25: Vývojové diagramy funkcie hľadania kandidátov

5.4.3.7 Implementácia pomocou CUDA

Pre zrýchlenie detekčného algoritmu, presunieme výpočet aktuálneho skóre danej detekcie na grafickú kartu. Aby sme mohli na zadaných dátach spustiť kernel, je nutné dáta najskôr presunúť do pamäte GPU. Do pamäte teda najskôr nakopírujeme

všetky šablóny objektu a aktuálnu snímku z kamery po predspracovaní. Na alokáciu pamäte na grafickej karte poskytuje CUDA funkciu *cudaMemcpy()*, ktorej zadáme cieľovú a zdrojovú adresu pamäte a počet bajtov, ktoré sa majú prekopírovať, podobne ako pri štandardnej funkcii *memcpy()*, ale musíme dodať ešte smer prenosu dát ako parameter do tejto funkcie. Podporované sú 3 režimy:

1. *cudaMemcpyHostToDevice* – kopírovanie zo systémovej RAM do pamäte GPU,
2. *cudaMemcpyDeviceToHost* – kopírovanie z GPU do systémovej RAM,
3. *cudaMemcpyDeviceToDevice* – kopírovanie v rámci pamäte GPU.

Keďže pri spúšťaní kernel funkcie, dostanú všetky spustené instance na GPU rovnaké argumenty, ktoré sme zadali vo volaní funkcie, musíme v kerneli na základe indexu vlákna a bloku určiť každému vláknu jeho index v rámci všetkých šablón objektu. Na odlíšenie spustených vláken v rámci GPU má CUDA definovaných niekoľko premenných, ktorých hodnoty môžeme použiť v kerneloch [32] :

- *gridDim* – táto premenná je typu *dim3* (reprezentuje 3D bod) a obsahuje veľkosti vytvorenej mriežky,
- *blockIdx* – premenná je typu *uint3* a obsahuje index bloku v rámci mriežky,
- *blockDim* – premenná je typu *dim3* a obsahuje veľkosti bloku,
- *threadIdx* – premenná je typu *uint3* a obsahuje index vlákna v rámci bloku,
- *warpSize* – táto premenná je typu *int* a obsahuje *warp* veľkosť vo vláknach, hodnota je daná hardvérovo a určuje počet vláken, ktoré zdieľajú rovnaký programový kód, v prípade modulu Jetson TX2 má hodnotu 32.

V našej kernel funkcii použijeme tieto premenné na určenie adresy šablóny, ktorá má byť daným vláknom vyhodnotená. Všetky uvedené premenné sú trojrozmerné a pristupovať k ich jednotlivým elementom *x*, *y* a *z* môžeme cez operátor prístupu. V našom programe použijeme tieto premenné ako jednorozmerné a používať budeme iba rozmer *x*. V nasledujúcom výpise je uvedený výpočet správneho ukazovateľa na požadovanú šablónu v pamäti GPU:

```
1 // Unique idx (template number)
2 const int idx = threadIdx.x + blockDim.x * blockIdx.x;
3
4 // Check if current index is not out of range
5 if (idx >= numTmpl)
6     return;
7
8 // Pointer to first template point
9 const int* currentTemplate = &templatePtr[2 * idx * maxTmplLength];
```

Výpis 6: Určenie ukazovateľa na požadovanú šablónu a ošetrovanie rozsahu

Premenná *templatePtr* vo výpise je argument predávaný kernelu pri jeho volaní a obsahuje adresu odkazujúcu na prvú šablónu. Ďalšími parametrami, ktoré sú kernelu pri volaní predávané je maximálna veľkosť jednej šablóny označená ako

maxTmplLength a pole s veľkosťami šablón, pretože každá šablóna môže mať inú veľkosť. Kvôli tejto vlastnosti bolo nutné šablóny v pamäti GPU zarovnať tak, aby sa dali vyššie uvedeným popisom indexovať. Pred kopírovaním šablón do pamäte GPU je najskôr stanovená dĺžka najväčšej šablóny. Následne je alokovaný jeden súvislý blok pamäte s veľkosťou *maxTmplLength * N*, kde *N* označuje počet šablón a jednotlivé šablóny sú potom do pamäte prekopírované. Takto zarovnané šablóny už môžeme prekopírovať do GPU a použiť popísané indexovanie.

Ďalšími dôležitými parametrami pri volaní kernelu sú parametre v trojitých zátvorkách určujúce počet instancií kernelu, ktoré budú spustené. Stanovenie počtu použitých vlákien a blokov je na programátorovi, ktorý daný kernel používa. Prvým krokom je stanoviť počet vlákien v bloku. Toto číslo musí byť v rozsahu 32 – 1024 a musí byť násobkom hodnoty *warp* popísanej vyššie. Optimálne hodnoty pre každú aplikáciu môžu byť iné a získať ich sa podarí väčšinou len experimentovaním. V našej aplikácii stanovíme počet vlákien 128, čo odpovedá hardvérovému usporiadaniu modulu Nvidia Jetson TX2 (jeden multiprocessor SM má veľkosť 128). Počet blokov, na ktorých budeme kernel spúšťať určíme tak, aby sa funkcia zavolała pre každú šablónu zadávanú do programu. Počet určíme nasledovným vzorcom:

$$\mathbf{blocks} = \mathbf{floor} \left(\frac{\mathbf{templates} + \mathbf{TpB} - 1}{\mathbf{TpB}} \right) \quad (2)$$

blocks – počet blokov pre kernel

templates – počet šablón používaných pri detekcii

TpB – stanovený počet vlákien na blok

floor() – funkcia, ktorá zaokrúhli výsledok nadol

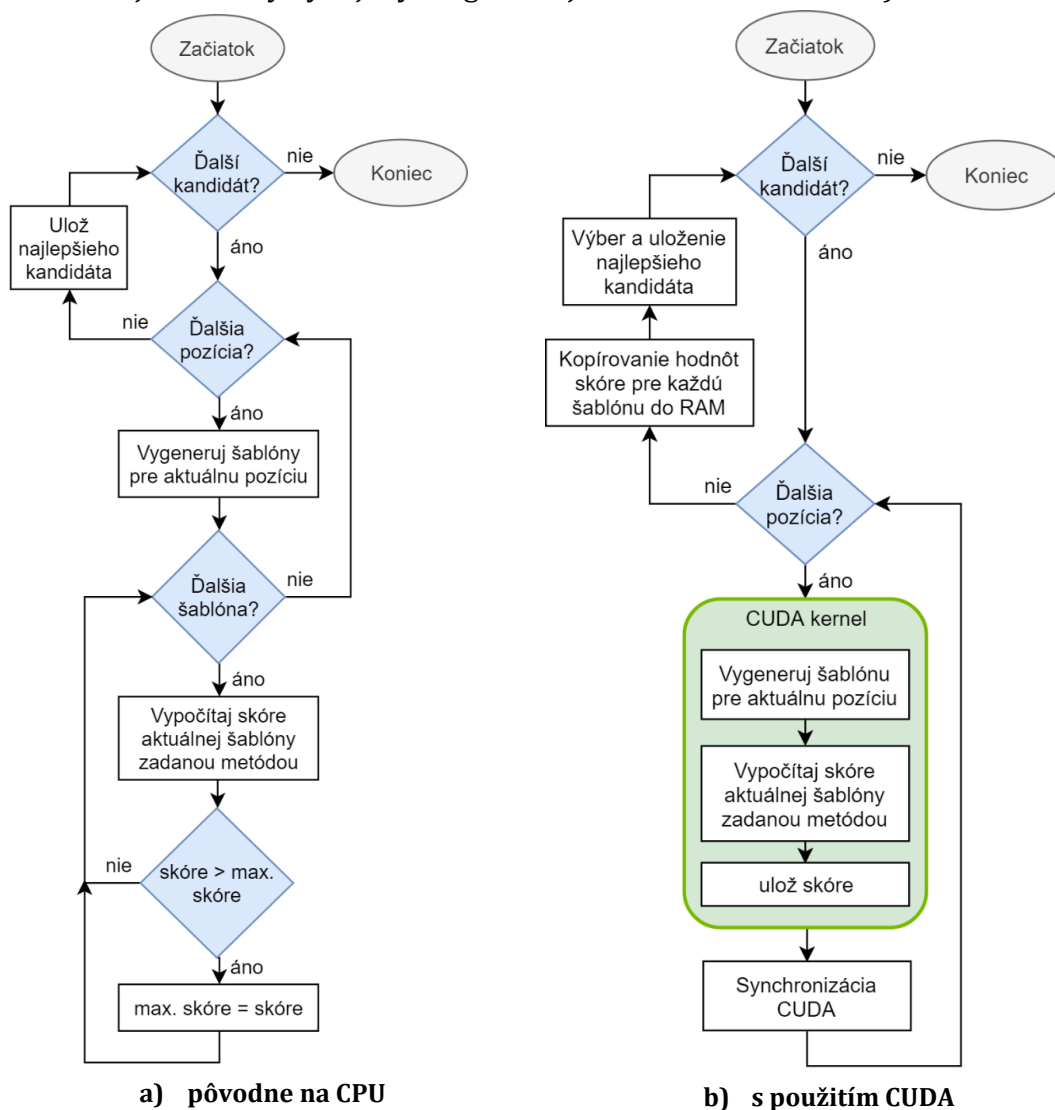
Po určení správnej adresy šablóny v kerneli nasleduje výpočet zhody s aktuálnou snímkou, ktorá je kernelu predaná podobne ako vyššie popísané premenné. Pri vytváraní kernelu, však nemôžeme zabudnúť na dôležité ošetrenie. Je to ošetrenie týkajúce sa indexu šablóny, s ktorou má kernel pracovať. Toto ošetrenie je nutné z dôvodu, že CUDA nedokáže spustiť iba časť vlákien bloku, ale ak daný blok spustíme, spustia sa na ňom všetky vlákna. Problém nastáva vo chvíli, keď počet šablón, nad ktorými výpočet spúšťame, nie je násobkom počtu vlákien v bloku, v našom prípade 128. Ak napríklad chceme spustiť kernel s 200 šablónami, musíme spustiť pri počte vlákien 128 na blok 2 bloky. Tým sa ale spustí celkovo 256 instancií kernelu, preto musíme indexy, ktoré presahujú počet šablón odfiltrovať kvôli prístupom do nealokovanej pamäte. Toto ošetrenie je vidieť v uvedenom výpise kódu na riadkoch 5 a 6, pričom premenná *numTmpl* je predávaná pri volaní a vyjadruje počet šablón objektu.

Vývojový diagram celej funkcie hľadania kandidátov detekcie je možné vidieť na Obr. 25 b). Dôležitou zmenou oproti pôvodnému vývojovému diagramu je blok synchronizácie CUDA. Ak v programe použijeme volanie kernelu, výpočet sa spustí

asynchrónne, a preto pred spracovaním výsledkov kernelu musíme najskôr počkať na dokončenie všetkých operácií. Na toto slúži funkcia *cudaDeviceSynchronize()*, ktorá zablokuje proces, kým sa operácie nedokončia. Po prejdení všetkých pozícií sa z GPU vyčítajú dosiahnuté skóre a vyberú sa najlepší kandidáti pre každú pozíciu. Týmto volanie funkcie končí.

5.4.3.8 Hľadanie presnej polohy objektu

Ďalším krokom v programe 3D detekcie je zo získaných kandidátov stanoviť presnú polohu objektu. Z kandidátov na detekciu, ktorých sme získali, vyberieme na základe skóre a pozície v obraze určitý počet, ktorý spĺňa požadované parametre a stanovený minimálny prah. Títo kandidáti budú teraz vstupom do ďalšej, výpočtovo náročnej funkcie upresnenia polohy kandidátov. Funkcia prechádza všetkých kandidátov a snaží sa v okolí kandidáta nájsť ešte lepšieho kandidáta. Pre lepšiu prehľadnosť je uvedený vývojový diagram tejto funkcie na Obr. 26 a).



Obr. 26: Vývojové diagramy funkcie upresnenia polohy

Z diagramu vidíme, že vo funkciách sú 3 slučky, ktoré zabezpečujú prechod programu cez všetkých kandidátov, všetky pozície a všetky šablóny. Náročnosť tejto funkcie je veľmi závislá na počte vybraných kandidátov a tiež na veľkosti okolia, ktoré sa pri hľadaní používa. V tejto funkcii, podobne ako to bolo pri predchádzajúcej funkcii, sa nám ako najjednoduchšie paralelizovanie ponúka slučka, ktorá prechádza všetky šablóny. Po podobných úpravách ako v predchádzajúcej funkcii dostaneme diagram zobrazený na Obr. 26 b).

5.4.3.9 Porovnanie rýchlosti detekcie

S upraveným programom s podporou CUDA sa pri detekcii dosiahlo pri určitých parametroch zlepšenie a pri iných zhoršenie, v porovnaní s výpočtom spusteným na procesore modulu Jetson TX2. Zlepšenie závisí od počtu šablón objektu. Použitie CUDA pri detekcii má zmysel až od určitého počtu šablón objektu. Kompletné porovnanie rýchlostí je uvedené v nasledujúcej kapitole 6. Dosiahnuté výsledky.

5.4.4 Oživovanie kamery

V kapitole venujúcej sa návrhu systému sme vybrali kameru od firmy FLIR. Pre používanie kamery FLIR vo vlastnej aplikácii poskytuje spoločnosť vlastnú SDK knižnicu s názvom Spinnaker. V čase implementácie knižnice bola dostupná pre modul Jetson TX2 verzia SDK 1.12. Pri oživovaní vybratej kamery FLIR sa vyskytli problémy. Aj napriek veľkému úsiliu sa na module Jetson TX2 nepodarilo prenos z tejto kamery rozbehnúť spoľahlivo. Prenos z kamery, ktorý sme skúšali na priložených vzorových aplikáciách, nefungoval stabilne a snímku z kamery sa nepodarilo vyčítať vždy. Z desiatich snímok, ktoré testovací program vyčítaval, sa úspešne podarilo vyčítať približne polovicu a tento počet sa náhodne menil pri opakovaní. Uvedené problémy boli pravdepodobne spôsobené použitím staršej verzie SDK. Na klientskom počítači s operačným systémom Windows bola dostupná novšia verzia knižnice vo verzii 1.20, na ktorej prenos fungoval bez problémov. Kvôli uvedenému problému sme sa rozhodli implementáciu tejto kamery odložiť, kým výrobca nevydá pre modul TX2 novú verziu knižnice, ktorá odstráni uvedené nedostatky.

Namiesto kamery FLIR sme sa rozhodli pre testovacie účely použiť iný typ kamery, tentokrát s rozhraním USB. Na testovacie účely sme použili kameru HD Pro Webcam C920 od spoločnosti Logitech, ktorej vlastnosti boli pre naše použitie vyhovujúce. Základné parametre kamery:

- Rozhranie: USB 2.0,
- Automatické zaostrovanie,
- Rozlíšenie: 1920 x 1080,
- Max. snímková frekvencia: 30,

- Cena: 1750+ Kč (www.heureka.cz, 4.5.2019).

Oživenie tejto kamery v našej aplikácii bolo jednoduché, keďže ovládače ku kamere si operačný systém doinštaloval automaticky a knižnica OpenCV sa k nej mohla pripojiť bez ďalšieho nastavovania.

5.5 Ovládacia aplikácia

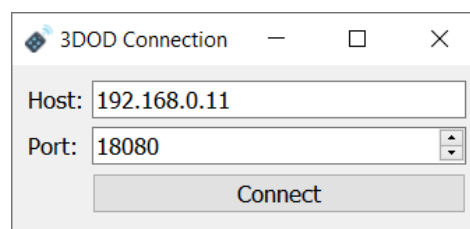
Na ovládanie hlavnej aplikácie, ktorá beží na module Jetson TX2 bola vytvorená klientská aplikácia. Pomocou tejto aplikácie sa môžeme k danému modulu na lokálnej sieti pripojiť a plne ovládať detekčný program.

5.5.1 Qt

Pre vytvorenie grafického rozhrania aplikácie sme použili platformu Qt. Qt je multiplatformový softvérový framework, ktorý zahŕňa v sebe množstvo C++ knižníc a kompletné integrované vývojové prostredie. Qt tiež ponúka plugin do vývojového prostredia Microsoft Visual Studio, ktoré budeme v práci používať. Vytváranie grafických aplikácií v Qt je veľmi jednoduché a intuitívne, vďaka veľkému množstvu preddefinovaných UI elementov.

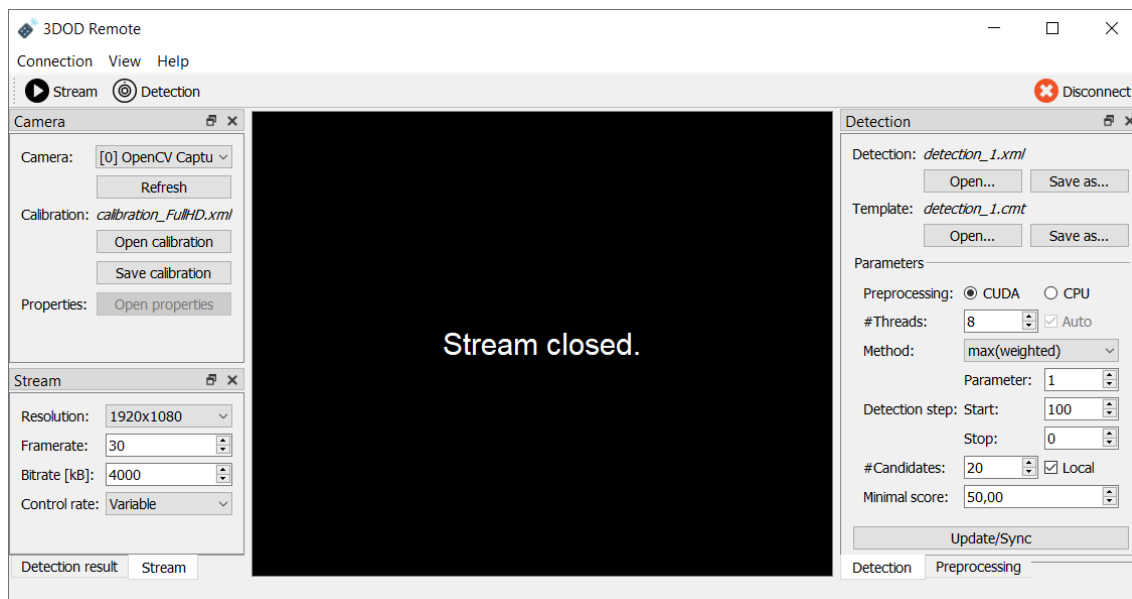
5.5.2 Popis aplikácie

Cez túto aplikáciu je možné sa pripojiť vzdialene k hlavnej aplikácii, ktorá beží na module Jetson TX2. Aplikácia na pripojenie vyžaduje zadanie IP adresy a portu, na ktorom je server v module spustený. Po spustení klientskej aplikácie sa zobrazí nasledovná výzva na pripojenie, kde potrebné údaje zadáme a nadviaže sa TCP spojenie.



Obr. 27: Výzva klientskej aplikácie na pripojenie

Po úspešnom pripojení sa nám zobrazí hlavné okno aplikácie, pomocou ktorého môžeme beh programu v module Jetson ovládať a konfigurovať. V tomto okne si tiež môžeme zobrazíť obraz použitej kamery v module. Hlavné okno aplikácie je zobrazené na nasledovnom obrázku a pozostáva z niekoľkých častí. Jednotlivé časti si popíšeme na uvedenom obrázku.



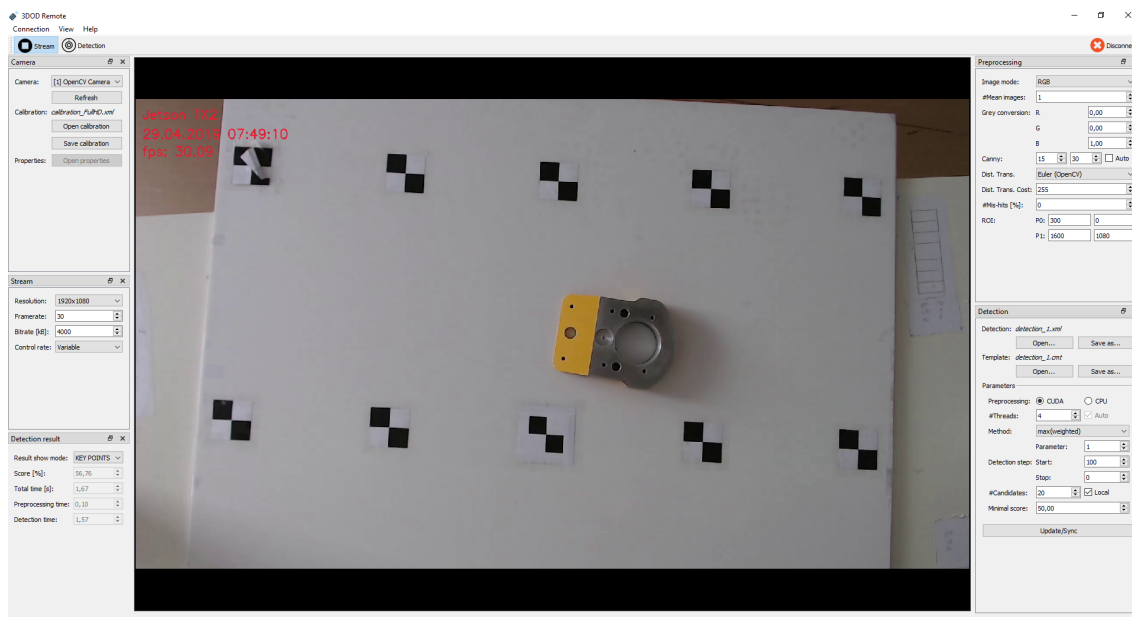
Obr. 28: Hlavné okno ovládacej klientskej aplikácie

Okno pozostáva z niekoľkých grafických modulov, ktorými môžeme hlavný program v module Jetson ovládať. Tieto grafické moduly je možné zobraziť alebo skryť podľa potreby užívateľa. Ďalej je v okne zobrazený náhľad zvolenej kamery (na zobrazenom obrázku nie je táto funkcia zapnutá). Pre zapnutie tejto funkcie musíme najskôr vybrať požadovanú kameru zo zoznamu v záložke *Camera*. Ďalej v záložke *Stream* môžeme definovať parametre prenosu. Nastaviteľné sú tieto parametre:

- rozlíšenie prenášaného obrazu,
- snímková frekvencia,
- bitová rýchlosť, ktorá ovplyvňuje kvalitu prenosu pomocou zvolenej maximálnej šírky pásma,
- typ bitového prenosu.

Tieto nastavenia boli v programe ponechané pre testovacie účely a tiež pre možnosť odľahčiť procesor ovládaného modulu znížením rozlíšenia alebo frekvencie, pretože kompresia obrazu, ktorú modul robí, významne zaťažuje procesor, a tak odoberá výpočtový výkon potrebný na detekciu objektov hlavným programom.

Po nastavení požadovaných parametrov prenosu môžeme prenos obrazu spustiť tlačidlom *Stream* v hornej lište. Po stlačení tlačidla dôjde k odoslaniu požiadavky hlavnému programu na pripojenie k vybranej kamere a spusteniu streamu. Zároveň klientská aplikácia otvorí potrebný UDP port a čaká na príjem obrazu. Po úspešnom pripojení sa po chvíli v okne zobrazí obraz kamery. Aplikáciu so zapnutým streamom je možné vidieť na nasledujúcom obrázku.



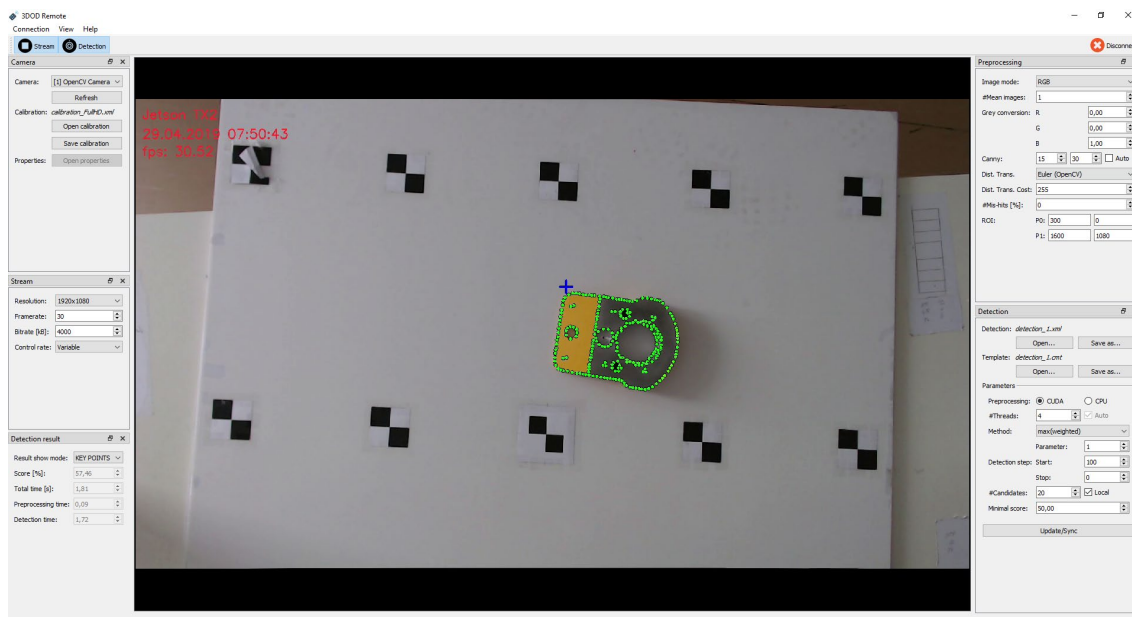
Obr. 29: Hlavné okno ovládacej klientskej aplikácie so zapnutým streamom

Pred spustením detekčného programu je nutné modulu Jetson TX2 poslať kalibračné údaje použitej kamery, ak už neboli načítané. Toto vykonáme tlačidlom *Open calibration* v záložke *Camera*. V zobrazenom okne vyberieme kalibračný súbor k danej kamere a voľbu potvrdíme. Vybraný súbor je potom pomocou TCP spojenia prenesený do modulu Jetson a následne načítaný.

Posledným krokom pred spustením detekčného programu je nahratie samotných šablón objektu, ktorý chceme detegovať a nastavenie parametrov detekcie. Šablóny objektu a parametre detekcie nahráme do modulu, podobne ako pri kalibračnom súbore kamery použitím tlačidiel v záložke *Detection*. O tom, či sú v module načítané potrebné súbory pre detekciu, nás aplikácia informuje zobrazením príslušného názvu súboru v danej záložke.

Po nahratí všetkých súborov je program pripravený a môžeme spustiť detekciu objektov príslušným ovládacím tlačidlom v hornej lište aplikácie. Po zapnutí detekcie program začne v slučke vyhľadávať zadaný objekt v obraze a výsledky detekcie sa začnú zobrazovať v náhlade kamery. Spôsob zobrazenia detegovaného objektu je možné vidieť na obrázku na nasledujúcej strane.

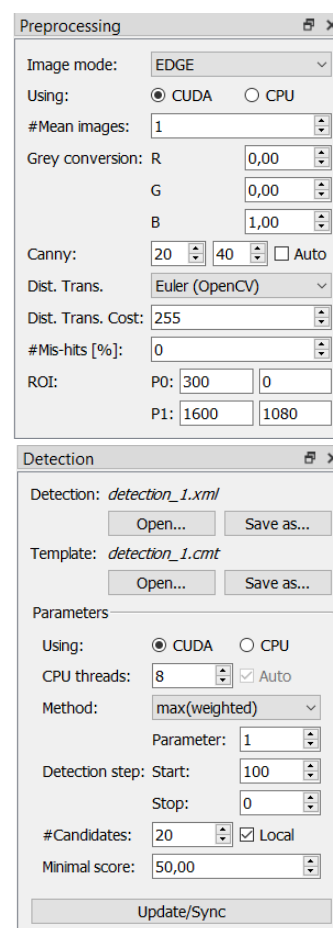
Ďalej je možné v záložke *Detection result* vidieť výsledky detekcie. Zobrazená je percentuálna zhoda, ktorú sa podarilo dosiahnuť a časy jednotlivých krokov detekcie, spolu s výsledným časom. Samotná zistená pozícia je reprezentovaná transformačnou maticou, ktorá nie je v aplikácii zobrazená. Táto matica sa v reálnej aplikácii predá ďalšiemu nadradenému systému, ktorý následne túto informáciu môže využiť. Môže to byť napr. robot, ktorý na základe zistenej polohy objekt uchopí.



Obr. 30: Hlavné okno ovládacej klientskej aplikácie so zapnutou detekciou

Parametre detekcie a predspracovania je možné nastavovať za behu aplikácie v záložkách *Preprocessing* a *Detection*. V prvej záložke nastavujeme parametre týkajúce sa spracovania snímky z kamery. Môžeme tu nastaviť počet priemerovaných obrázkov, vlastné váhy pre konverziu z RGB do odtieňov šedej a ďalšie nastavenia týkajúce sa predspracovania. Dôležitým nastavením, ktoré bolo pridané vytvorenou aplikáciou, je možnosť zvoliť, či predspracovanie prebehne na CPU alebo s podporou CUDA. Aby sme mohli možnosť CUDA zvoliť, musí byť knižnica OpenCV zostavená s podporou CUDA, inak prepnutie tohto nastavenia neovplyvní aplikáciu v module Jetson TX2.

V záložke *Detection* môžeme meniť parametre detekčného algoritmu. Podobne ako pri predspracovaní aj k týmto nastaveniam bola pridaná možnosť zvoliť, ako prebehne výpočet (CPU alebo CUDA). Pri detekcii je tiež možnosť zvoliť si pri spustení cez CPU počet vlákien, medzi ktoré sa výpočtové operácie rozdelia. Parametre, ktoré v aplikácii zmeníme, musíme potvrdiť tlačidlom *Update/Sync*. Takto sa nastavenia uložia a nahrajú do modulu, kde sa v nasledujúcej iterácii detekčného programu aplikujú.



Obr. 31: Záložky aplikácie *Preprocessing* a *Detection*

6 DOSIAHNUTÉ VÝSLEDKY

V nasledujúcej kapitole sú popísané dosiahnuté výsledky zrýchlenia poskytnutého detekčného programu. Všetky uvedené časy v tabuľkách boli získané ako priemer z desiatich meraní, aby sa vylúčilo kolísanie hodnôt vplyvom zaťaženia CPU a GPU inými procesmi. Všetky merania boli merané za rovnakých podmienok v rovnakom prostredí, s použitím popísanej testovacej kamery Logitech, bez zapnutého streamovania.

6.1 Porovnanie časov detekčného algoritmu

Meranie času detekčného algoritmu prebehlo na poskytnutých šablónach objektu s veľkosťami 37, 73, 181, 361, 2896 a 6859.

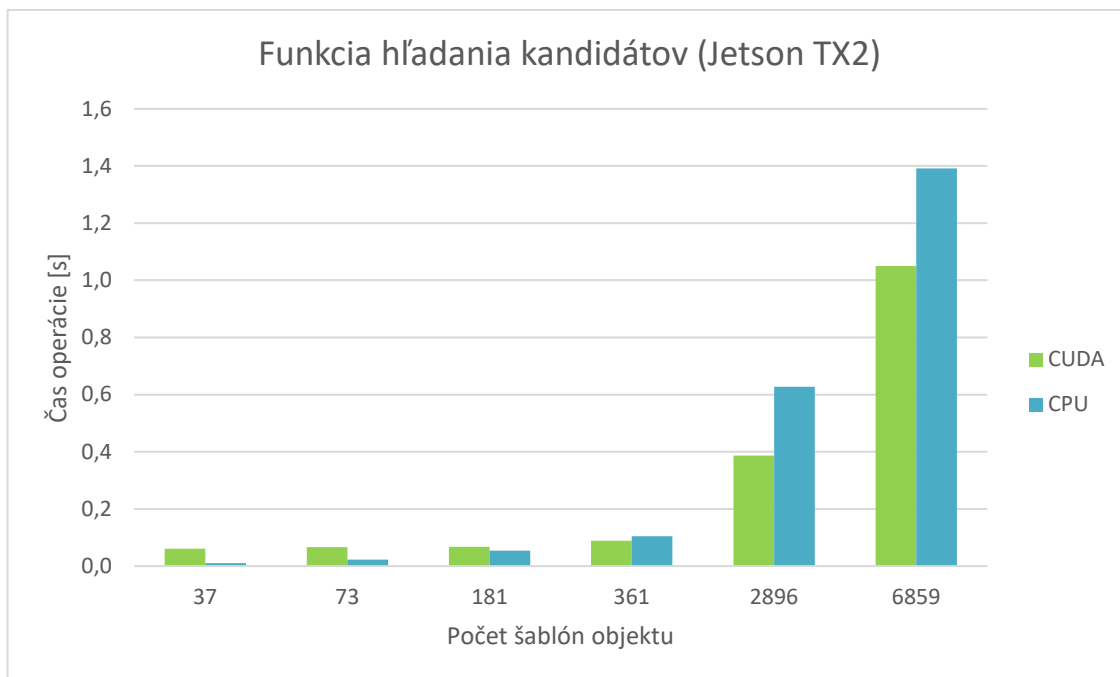
6.1.1 Výsledky modulu Jetson TX2

Meranie bolo najskôr vykonané na module Jetson TX2 v móde maximálneho výkonu procesora (mód 0). V nasledujúcej tabuľke sú uvedené časy oboch upravovaných funkcií a celkový čas detekčného programu s použitím výpočtu CUDA a CPU.

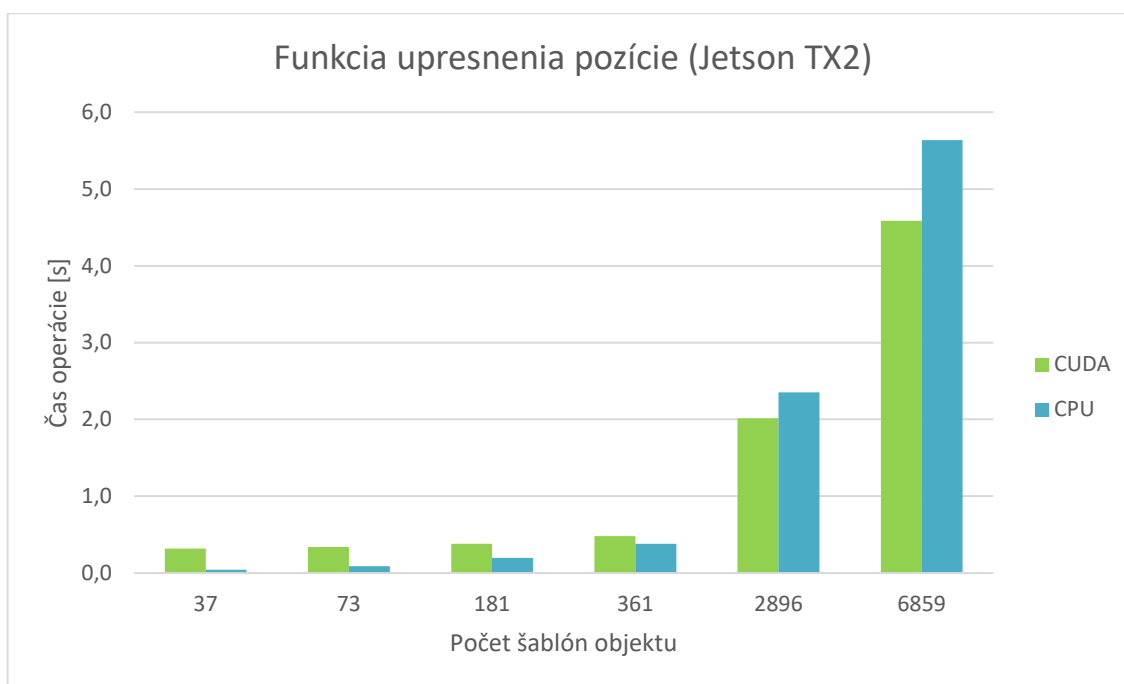
Počet šablón	CUDA			CPU		
	Hľadanie kandidátov [ms]	Upresnenie pozície [ms]	Celkový čas [ms]	Hľadanie kandidátov [ms]	Upresnenie pozície [ms]	Celkový čas [ms]
37	61	316	1131	11	42	1000
73	66	338	1150	23	89	1069
181	67	379	1197	54	197	1171
361	88	481	1311	105	381	1421
2896	386	2012	3087	628	2354	3693
6859	1049	4585	6256	1392	5637	7713

Tab. 5: Namerané časy detekčného programu na Jetson TX2

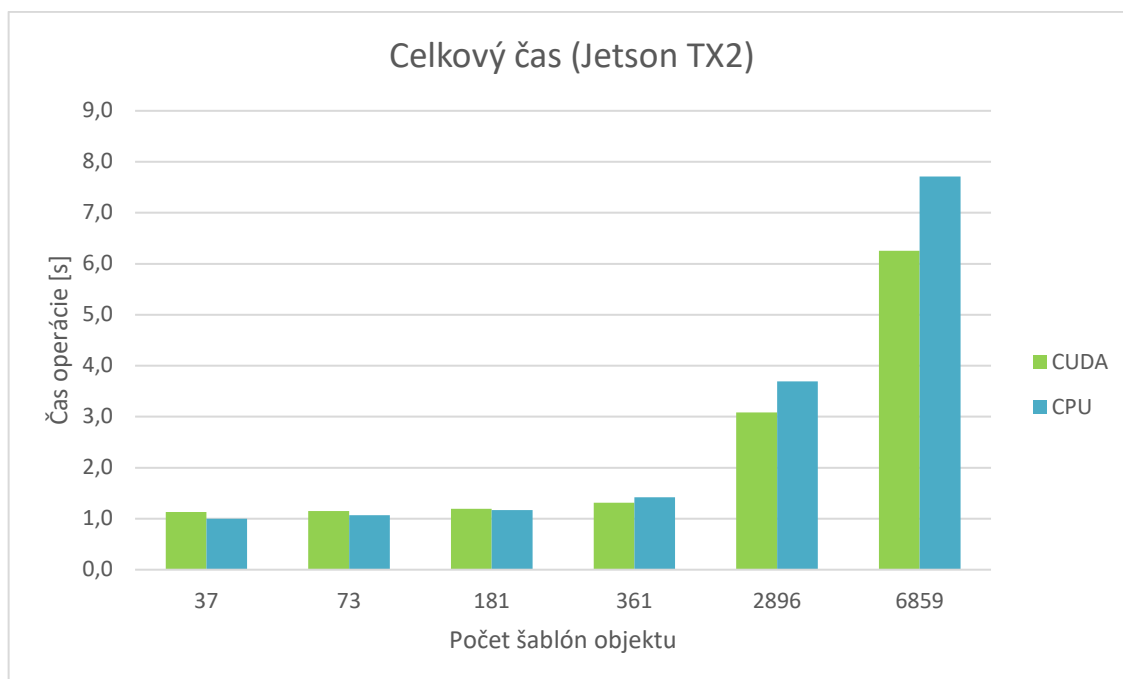
Na nasledujúcich stranách sú zobrazené hodnoty z uvedenej tabuľky v prehľadných grafoch. Grafy zobrazujú závislosť času danej funkcie a počtu použitých šablón objektu. V grafoch sú zobrazené časy operácií pri použití CPU aj CUDA variant funkcií. Prvý graf znázorňuje závislosť funkcie hľadania kandidátov. Druhý graf znázorňuje závislosť funkcie upresnenia pozície a posledný ukazuje závislosť celkového času detekčného algoritmu. Celkový čas detekcie zahŕňa čas predspracovania obrazu, časy oboch funkcií a ďalšie operácie, ktorá sa musia vykonať sériovo.



Graf 1: Čas funkcie hľadania kandidátov v závislosti od počtu šablón objektu



Graf 2: Čas funkcie upresnenia pozície v závislosti od počtu šablón objektu



Graf 3: Celkový čas detekčného algoritmu v závislosti od počtu šablón objektu

Z uvedených grafov vidíme, že použitie technológie CUDA sa oplatí ak počet šablón objektu používaných pri detekcii dosiahne určitý počet. V našom prípade nastáva zlom pri použití 181 šablón, kde sa časy pri použití CPU a CUDA približne rovnajú. S narastajúcim počtom šablón už prestáva prevažovať čas réžie GPU, využíva sa plná kapacita GPU a prevažuje užitočný výpočtový čas, čím sa dosahuje celkový čas nižší než na CPU. Pri počte šablón 6859 sa tak podarilo dosiahnuť zlepšenie celkového času o približne 19%.

6.1.2 Výsledky testovacieho počítača

V rovnakých podmienkach ako modul Jetson TX2 bolo vykonané meranie aj na klasickom počítači. Základné komponenty použitého počítača:

- Procesor: Intel Core i7-7700HQ (3.80 GHz, max. 8 vláken),
- GPU: Nvidia GTX 1050,
- RAM: 16GB 2400 MHz.

Pre porovnanie parametrov počítača a modulu si uvedieme tabuľku s parametrami jednotlivých grafických jednotiek. Zobrazené sú len niektoré parametre s rozdielnymi hodnotami.

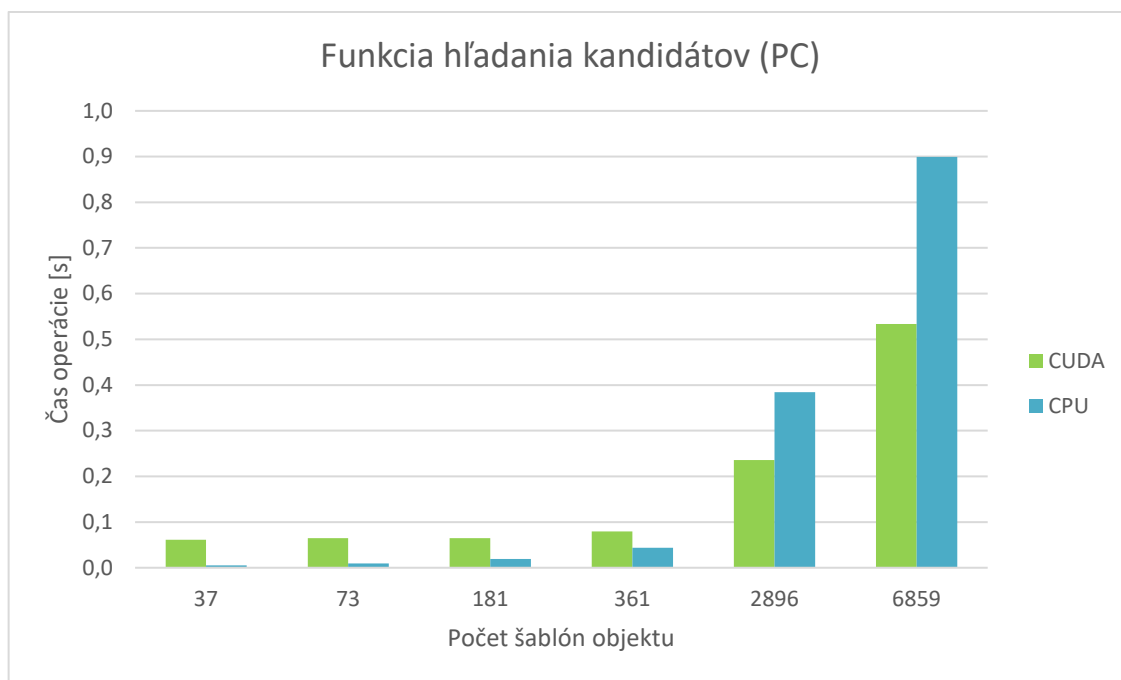
	Nvidia GTX 1050	Nvidia Tegra X2
CUDA jadier	640	256
Počet SM multiprocesorov	5	2
Globálna pamäť	4096 MB	7850 MB
Takt GPU	1,49 GHz	1,3 GHz
Takt pamäte	1600 MHz	3504 MHz

Tab. 6: Porovnanie parametrov grafických jednotiek PC a modulu Jetson TX2

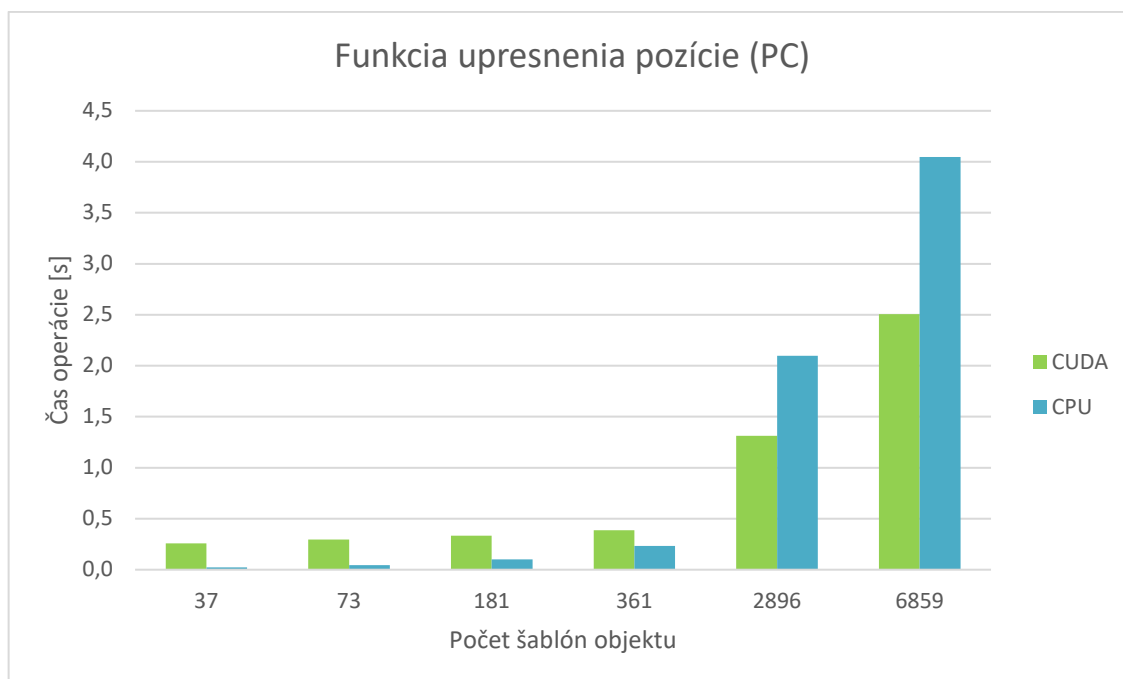
Dosiahnuté výsledky merania na testovanom počítači sú uvedené v nasledujúcej tabuľke a grafoch zobrazujúcich rovnaké závislosti ako pri hodnotách získaných na module Jetson TX2.

Počet šablón	CUDA			CPU		
	Hľadanie kandidátov [ms]	Upresnenie pozície [ms]	Celkový čas [ms]	Hľadanie kandidátov [ms]	Upresnenie pozície [ms]	Celkový čas [ms]
37	61	260	636	5	23	401
73	65	296	671	9	46	425
181	64	333	705	20	102	497
361	80	387	777	44	232	647
2896	235	1313	1864	385	2098	2865
6859	534	2505	3313	900	4048	5279

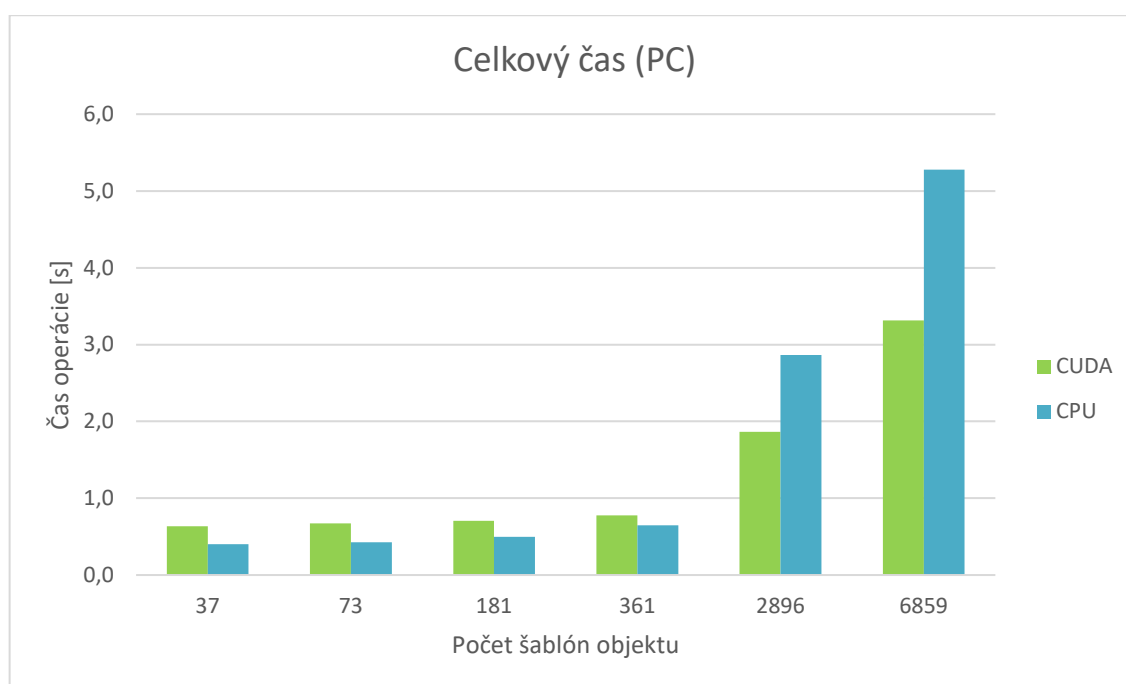
Tab. 7: Namerané časy detekčného programu na testovanom PC



Graf 4: Čas funkcie hľadania kandidátov v závislosti od počtu šablón objektu



Graf 5: Čas funkcie upresnenia pozície v závislosti od počtu šablón objektu



Graf 6: Celkový čas detekčného algoritmu v závislosti od počtu šablón objektu

Z uvedených grafov vidíme, že na testovanom počítači sa rozdiel medzi GPU a CPU prejavil výraznejšie. Rýchlejšie spracovanie sa dosiahlo na počítači až pri použití šablón s počtom 2896. Pri počte šablón 6859 sa podarilo dosiahnuť celkové zlepšenie času detekcie približne o 37%.

ZÁVER

Úlohou práce bolo navrhnúť systém počítačového videnia na platforme Nvidia s operačným systémom Linux. V práci bola najskôr vykonaná rešerš vhodných zariadení Nvidia na trhu a bolo vybraté najoptimálnejšie riešenie. Ako platforma systému bol vybratý modul Nvidia Jetson TX2. Jedná sa o miniatúrny počítač s veľkosťou mobilného telefónu, ale s vysokým výkonom vďaka procesorovému komplexu s dvoja procesormi a grafickým čipom s 256 CUDA jadrami.

V práci bol ďalej riešený spôsob komunikácie medzi nadradeným počítačom, ktorým sa dá navrhnutý systém nastavovať a ovládať. Bolo nutné navrhnúť spôsob, akým budú tieto zariadenia navzájom komunikovať. Pre riadiace príkazy a posielanie konfiguračných súborov do systému bol vytvorený komunikačný kanál s protokolom TCP. Tento protokol bol vybratý kvôli svojim vlastnostiam, najmä spoľahlivému prenosu. Ďalej bolo nutné navrhnúť spôsob, akým preniesť obraz kamery do vzdialeného počítača. Na tento účel bol vytvorený ďalší komunikačný kanál s protokolom UDP. Tento protokol bol vybratý kvôli svojej nízkej odozve. Prípadné straty paketov, ktoré môžu pri tomto protokole nastať nie sú problémom, keďže obraz je použitý len na vizualizáciu.

Ďalej bola navrhnutá bloková schéma celého systému a spôsob komunikácie medzi jednotlivými blokmi. Boli vybraté jednotlivé komponenty systému a popísané ich vlastnosti. Kamera od firmy FLIR, ktorá bola pôvodne vybratá pre navrhnutý systém sa pri realizácii nepoužila kvôli nespoľahlivému prenosu z kamery. Toto mohlo byť pravdepodobne spôsobené použitím neaktuálnej knižnice kamery pre systém Linux na module Jetson TX2. Namiesto nej bola použitá na testovanie kamera Logitech, ktorá dosahovala požadovaných vlastností.

Na základe navrhutej schémy sme systém zrealizovali a vytvorili softvérové vybavenie pre modul Jetson TX2 a nadradený počítač. Na zabezpečenie komunikácie medzi oboma vytvorenými aplikáciami bola najskôr navrhnutá komunikačná knižnica s použitím protokolu TCP. Následne bol vytvorený UDP komunikačný kanál pre zobrazovanie náhľadu kamery v ovládacej aplikácii. Na implementáciu sme použili knižnicu OpenCV s pridaným modulom GStreamer.

Práca sa ďalej venovala úprave a optimalizácii detekčného algoritmu knižnice. Na zrýchlenie výpočtov bola použitá technológia CUDA od spoločnosti Nvidia. CUDA bola využitá dvoma spôsobmi. Prvým spôsobom bolo využitie CUDA funkcií v knižnici OpenCV. Pôvodný reťazec predspracovania obrazu vykonávaný CPU bol takýmto spôsobom prepísaný pre beh na GPU. Týmto sa dosiahlo zvýšenie rýchlosti predspracovania obrazu o približne 26%.

Druhým spôsobom bola implementácia výpočtovo náročných operácií detekčného programu pomocou CUDA kernel funkcií. Nahradené boli dve funkcie

detekčného algoritmu: hľadanie kandidátov a upresnenie pozície. Pôvodné funkcie, ktoré obsahovali slučky, boli nahradené volaním CUDA kernel funkcií, ktoré potrebné operácie spracovali paralelne na GPU. Touto úpravou sme dosiahli zlepšenie až od určitého množstva použitých šablón objektu kvôli réžii GPU. Pri maximálnom testovanom počte šablón sa tak podarilo dosiahnuť na module Jetson TX2 zlepšenie o približne 19% a na klasickom počítači s výkonnejšou grafickou jednotkou až 37%.

V rámci práce boli vytvorené dve softvérové aplikácie. Hlavná aplikácia pre modul Jetson TX2, ktorej hlavným komponentom je detekčná knižnica. Hlavnou úlohou aplikácie je určiť polohu hľadaného objektu v 3D súradniciach na základe obrazu kamery. Túto informáciu potom môže využiť nadradený systém. Druhá aplikácia slúži na ovládanie modulu Jetson TX2 a nastavenie detekčného programu.

Presunom aplikácie na modul Jetson TX2 s podporou CUDA sa tak podarilo vytvoriť miniatúrny systém s výkonom porovnateľným s testovaným počítačom. Celý systém bude v budúcnosti presunutý do jedného šasi, čím vznikne vstavaný inteligentný systém videnia. Takéto zariadenie s uvedeným softvérovým vybavením potom môže nájsť uplatnenie napr. v priemysle pri navádzaní robotov na uchopovanie objektov, pri zváraní a iných aplikáciách.

Literatúra

- [1] Silicon Highway LTD . *NVIDIA Jetson TX2 Module* [online]. [cit. 2018-12-04]. Dostupné z: <https://www.siliconhighwaydirect.co.uk/product-p/900-83310-0001-000.html>
- [2] Rozhranie USB 3.0. In: Wikipedia: the free encyclopedia [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2018-12-04]. Dostupné z: https://en.wikipedia.org/wiki/USB_3.0
- [3] NVIDIA Corporation. *NVIDIA Jetson systems* [online]. [cit. 2018-12-14]. Dostupné z: <https://www.nvidia.com/en-us/autonomous-machines/embedded-systems-dev-kits-modules/>
- [4] NVIDIA Corporation. *Jetson TX2 Module* [online]. [cit. 2018-12-14]. Dostupné z: <https://developer.nvidia.com/embedded/buy/jetson-tx2>
- [5] *Silicon Highway* [online]. Dostupné z: <https://www.siliconhighwaydirect.co.uk/>
- [6] NVIDIA Corporation. *NVIDIA Jetson TX2 Delivers Twice the Intelligence to the Edge* [online]. [cit. 2018-12-14]. Dostupné z: <https://devblogs.nvidia.com/jetson-tx2-delivers-twice-intelligence-edge/>
- [7] Your Cable Store Inc. *USB Cable Length Limitations* [online]. [cit. 2018-12-14]. Dostupné z: http://www.yourcablestore.com/USB-Cable-Length-Limitations-And-How-To-Break-Them_ep_42-1.html
- [8] How-To Geek, LLC. *Not All Ethernet Cables Are Equal* [online]. [cit. 2018-12-14]. Dostupné z: <https://www.howtogeek.com/210326/not-all-ethernet-cables-are-equal-you-can-get-faster-lan-speeds-by-upgrading/>
- [9] GigE Vision: In: Wikipedia: the free encyclopedia [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2018-12-20]. Dostupné z: https://en.wikipedia.org/wiki/GigE_Vision
- [10] MORAN, Peter. *CSI Cameras on the TX2 (The Easy Way)* [online]. [cit. 2018-12-20]. Dostupné z: <http://petermoran.org/csi-cameras-on-tx2/>
- [11] MIPI Alliance, Inc. *MIPI Camera Serial Interface 2 (MIPI CSI-2)* [online]. [cit. 2018-12-20]. Dostupné z: <https://mipi.org/specifications/csi-2>
- [12] Computer vision. In: Wikipedia: the free encyclopedia [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2018-12-20]. Dostupné z: https://en.wikipedia.org/wiki/Computer_vision
- [13] NVIDIA Corporation. *CUDA Toolkit* [online]. [cit. 2018-12-20]. Dostupné z: <https://developer.nvidia.com/cuda-toolkit>
- [14] NVIDIA Corporation. *What si CUDA?* [online]. [cit. 2018-12-20]. Dostupné z: https://www.nvidia.com/object/io_69526.html

- [15] NVIDIA Corporation. *About CUDA* [online]. [cit. 2018-12-20]. Dostupné z: <https://developer.nvidia.com/about-cuda>
- [16] NVIDIA Corporation. *What is CUDA?* [online]. [cit. 2018-12-20]. Dostupné z: <https://blogs.nvidia.com/blog/2012/09/10/what-is-cuda-2/>
- [17] NVIDIA Corporation. *JetPack* [online]. [cit. 2018-12-20]. Dostupné z: <https://developer.nvidia.com/embedded/jetpack>
- [18] OpenCV team. *About OpenCV* [online]. [cit. 2018-12-20]. Dostupné z: <https://opencv.org/about.html>
- [19] Transmission Control Protocol. In: Wikipedia: the free encyclopedia [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2018-12-20]. Dostupné z: https://en.wikipedia.org/wiki/Transmission_Control_Protocol
- [20] L DORDAL, Peter. *TCP Transport* [online]. [cit. 2018-12-20]. Dostupné z: <https://intronetworks.cs.luc.edu/current/html/tcp.html>
- [21] User Datagram Protocol. In: Wikipedia: the free encyclopedia [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2018-12-20]. Dostupné z: https://en.wikipedia.org/wiki/User_Datagram_Protocol
- [22] think-async. *Asio C++ Library* [online]. [cit. 2018-12-20]. Dostupné z: <https://think-async.com/Asio/>
- [23] MPEG-4 AVC. In: Wikipedia: the free encyclopedia [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2018-12-29]. Dostupné z: https://en.wikipedia.org/wiki/H.264/MPEG-4_AVC
- [24] High Efficiency Video_Coding. In: Wikipedia: the free encyclopedia [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2018-12-29]. Dostupné z: https://en.wikipedia.org/wiki/High_Efficiency_Video_Coding
- [25] MulticoreWare Inc. *x265 HEVC Upgrade* [online]. [cit. 2018-12-20]. Dostupné z: <https://x265.com/>
- [26] PORUBSKÝ, Števo. *H.265 / HEVC – kodek, ktorý bude vládnuť svetu?* [online]. [cit. 2018-12-20]. Dostupné z: <https://techbox.dennikn.sk/temy/h-265-hevc-kodek-ktory-bude-vladnut-svetu/>
- [27] FFmpeg. *About FFmpeg* [online]. [cit. 2018-12-20]. Dostupné z: <https://www.ffmpeg.org/about.html>
- [28] Daily Automation, s.r.o. *01 – Strojové videnie: Úvod (Machine Vision)* [online]. [cit. 2018-12-20]. Dostupné z: <http://dailyautomation.sk/01-strojove-videnie-uvod/>
- [29] MROZ. *Materiály k predmetu* [online]. [cit. 2018-12-20]. Dostupné z: http://midas.uamt.feec.vutbr.cz/ROZ/roz_cz.php
- [30] *Katalógový list kamery FLIR Blackfly S BFS-PGE-31S4C-C* [online]. [cit. 2018-12-20]. Dostupné z: <https://www.ptgrey.com/support/downloads/10823>

- [31] Kitware. *About CMake* [online]. [cit. 2019-04-25]. Dostupné z: <https://cmake.org/>
- [32] NVIDIA Corporation. *CUDA C Programming Guide* [online]. [cit. 2019-04-26]. Dostupné z: <https://docs.nvidia.com/cuda/cuda-c-programming-guide/>
- [33] Mendelova univerzita v Brně. *Hashovací funkce* [online]. [cit. 2019-05-01]. Dostupné z: https://is.mendelu.cz/eknihovna/opory/zobraz_cast.pl?cast=7029
- [34] *C++ MD5 Function* [online]. [cit. 2019-05-01]. Dostupné z: <http://www.zedwood.com/article/cpp-md5-function>
- [35] NVIDIA Corporation. *Download and Install JetPack* [online]. [cit. 2019-05-02]. Dostupné z: <https://docs.nvidia.com/jetson/archives/jetpack-archived/jetpack-33/index.html#jetpack/3.3/install.htm>
- [36] GStreamer. *What is GStreamer?* [online]. [cit. 2019-05-02]. Dostupné z: <https://gstreamer.freedesktop.org/documentation/application-development/introduction/gstreamer.html>
- [37] VP9. In: Wikipedia: the free encyclopedia [online]. San Francisco (CA): Wikimedia Foundation, 2001- [online]. [cit. 2019-05-02]. Dostupné z: <https://en.wikipedia.org/wiki/VP9>
- [38] Kalicinski, Marcin. *Knižnica rapidxml* [online]. [cit. 2019-05-04]. Dostupné z: <http://rapidxml.sourceforge.net/>
- [39] Connect Tech Inc. *Orbitty Carrier for NVIDIA® Jetson™ TX2/TX2i/TX1* [online]. [cit. 2019-05-06]. Dostupné z: <http://connecttech.com/product/orbitty-carrier-for-nvidia-jetson-tx2-tx1/>
- [40] JetsonHacks. *NVPModel – NVIDIA Jetson TX2 Development Kit* [online]. [cit. 2019-05-07]. Dostupné z: <https://www.jetsonhacks.com/2017/03/25/nvpmodel-nvidia-jetson-tx2-development-kit/>
- [41] NVIDIA Corporation. *Accelerated GStreamer User Guide* [online]. [cit. 2019-05-02]. Dostupné z: https://developer.download.nvidia.com/embedded/L4T/r32_Release_v1.0/Docs/Accelerated_GStreamer_User_Guide.pdf

Zoznam skratiek

Slovenské a české skratky

FEKT	-	Fakulta elektrotechniky a komunikačních technologií
VUT	-	Vysoké učení technické v Brně
OS	-	Operačný systém

Anglické skratky

CPU	-	Central Processing Unit
GPU	-	Graphics Processing Unit
AI	-	Artificial Intelligence
SoC	-	System on a Chip
DMA	-	Direct Memory Access
LAN	-	Local Area Network
RAM	-	Random Access Memory
SDK	-	Software Development Kit
TCP	-	Transmission Control Protocol
UDP	-	User Datagram Protocol
PoE	-	Power Over Ethernet
OCR	-	Optical Character Recognition
UI	-	User Interface
SDK	-	Software development kit

Zoznam príloh

Príloha A – Prehľad parametrov dostupných zariadení platformy NVIDIA

Príloha B – Fotodokumentácia

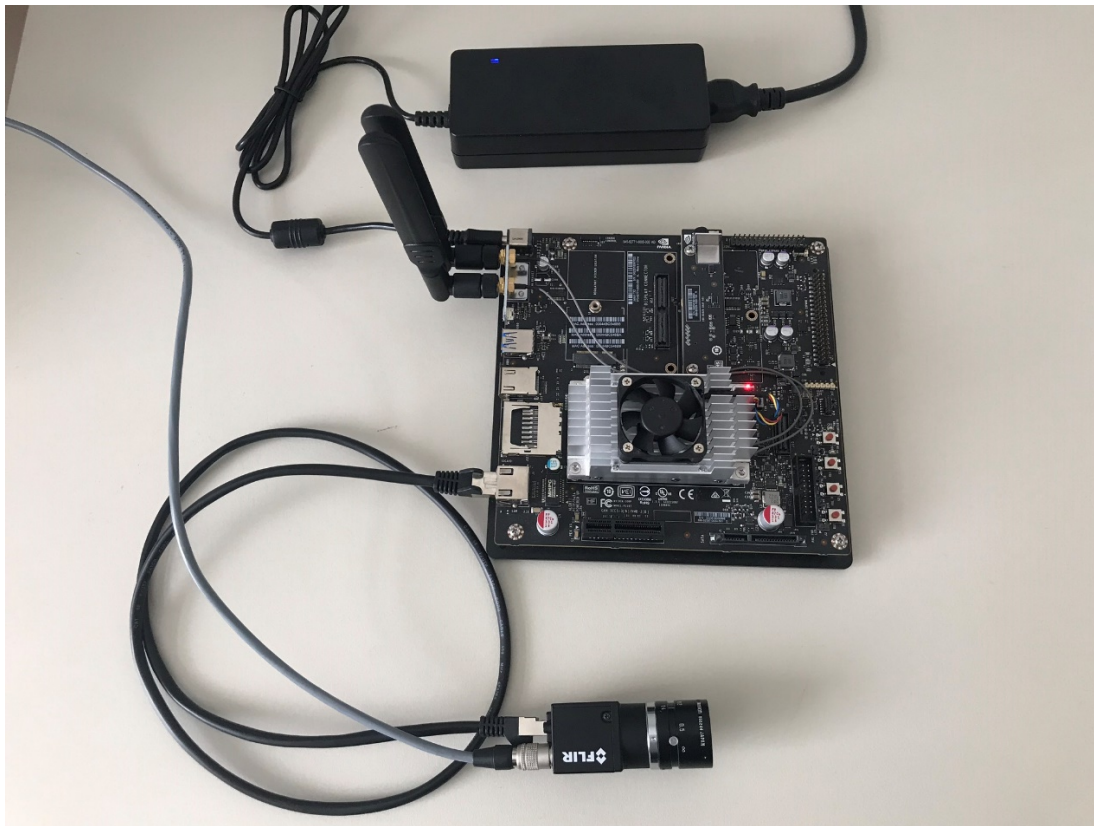
Príloha C – Obsah priloženého CD

Príloha A – Prehľad parametrov dostupných zariadení platformy NVIDIA

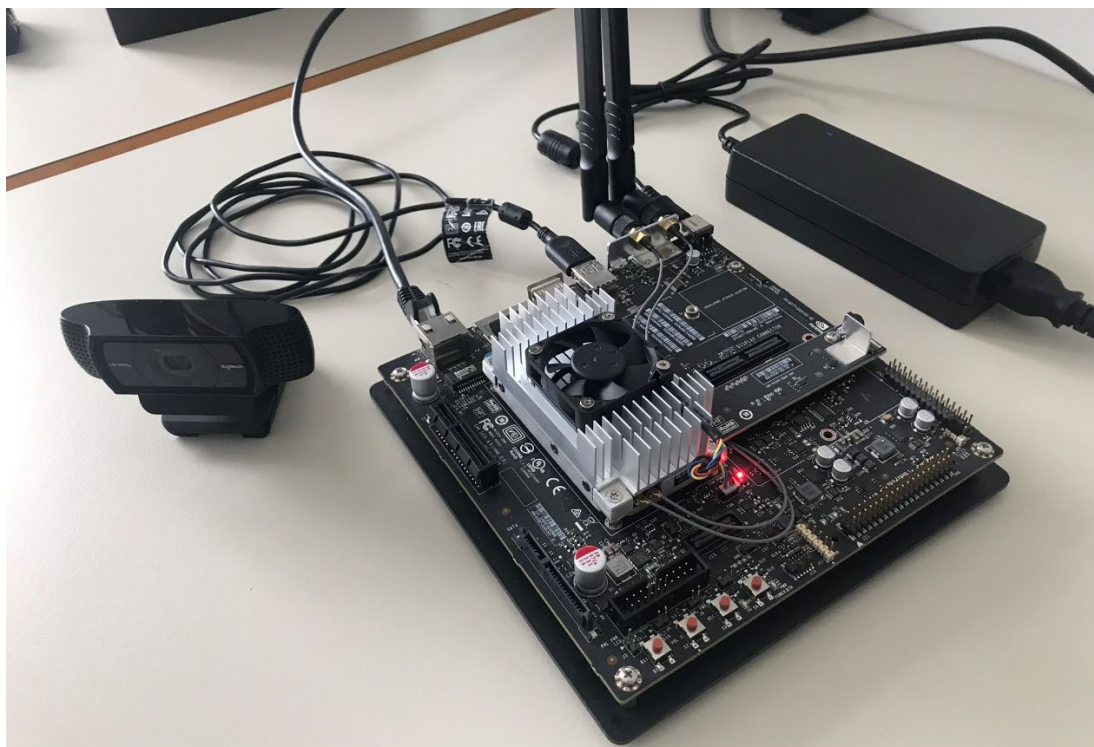
Produkt	Jetson AGX Xavier	Jetson TX2 Developer Kit	Jetson TX1 Developer Kit	Jetson TK1 Developer Kit
GPU	512-core Volta GPU with Tensor Cores	NVIDIA Pascal™, 256 CUDA cores	NVIDIA Maxwell™, 256 CUDA cores	NVIDIA Kepler™, 192 CUDA cores
CPU	8-core ARM v8.2 64-bit (8MB L2 + 4MB L3)	HMP Dual Denver 2 (2 MB L2) + Quad ARM® A57 (2 MB L2)	Quad ARM® A57 (2 MB L2)	NVIDIA 4-Plus-1 Quad ARM Cortex-A15
Video	(2x) 4Kp60 HEVC/(2x) 4Kp60 12-Bit Support Accelerator: 7-way VLIW Vision Processor	4K 60 Hz Encode (HEVC) 4K 60 Hz Decode (12-Bit Support)	4K 30 Hz Encode (HEVC) 4K 60 Hz Decode (10-Bit Support)	
Pamäť	16 GB 256-Bit LPDDR4x 137 GB/s	8 GB 128 bit LPDDR4 59.7 GB/s	4 GB 64 bit LPDDR4 25.6 GB/s	2 GB x 16 memory 64-bit
Displej	HDMI 2.0, eDP 1.2a, DP 1.4	2x DSI, 2x DP 1.2 / HDMI 2.0 / eDP 1.4	2x DSI, 1x eDP 1.4 / DP 1.2 / HDMI	HDMI
CSI	(16x) CSI-2 Lanes	Up to 6 Cameras (2 Lane) CSI2 D-PHY 1.2 (2.5 Gbps/Lane)	Up to 6 Cameras (2 Lane) CSI2 D-PHY 1.1 (1.5 Gbps/Lane)	
PCIe	x8 PCIe Gen4/x8 SLVS-EC, PCIe x1	Gen 2 1x4 + 1x1 OR 2x1 + 1x2	Gen 2 1x4 + 1x1	Half mini-PCIe slot 1
Úložisko	32GB eMMC 5.1, NVMe	32 GB eMMC, SDIO, SATA	16 GB eMMC, SDIO, SATA	16 GB 4.51 eMMC, SD/MMC, SATA
Ostatné	UART (for Wi-Fi/LTE) / I2S / PCM	CAN, UART, SPI, I2C, I2S, GPIOs	UART, SPI, I2C, I2S, GPIOs	RS232, DP/LVDS, Touch SPI 1x4 + 1x1 CSI-2, GPIOs, UART, HSIC, I2c
USB	USB 2.0, 2x Type C - USB 3.1, DP (Op.), PD (Op.)	USB 3.0, USB 2.0	USB 3.0, USB 2.0	USB 2.0 port, micro AB 1
Pripojenie	Gigabit Ethernet	Gigabit Ethernet 802.11ac WLAN Bluetooth	Gigabit Ethernet 802.11ac WLAN Bluetooth	Gigabit Ethernet
Rozmery	105 mm x 105 mm	50 mm x 87 mm	50 mm x 87 mm	127 mm x 127 mm x 26 mm
Výkon	20 - 30W	7.5 - 15W	7.5 - 15W	1 - 5W
Výpočtová schopnosť	7.2	6.2	5.3	3.2
Cena - modul*	nedostupné	£405.36	£303.24	nedostupné
Cena - kit*	£1,832.50	£441	£389.17	£140.15
Rok vydania	2018	2017	2015	2014

Príloha B – Fotografie práce

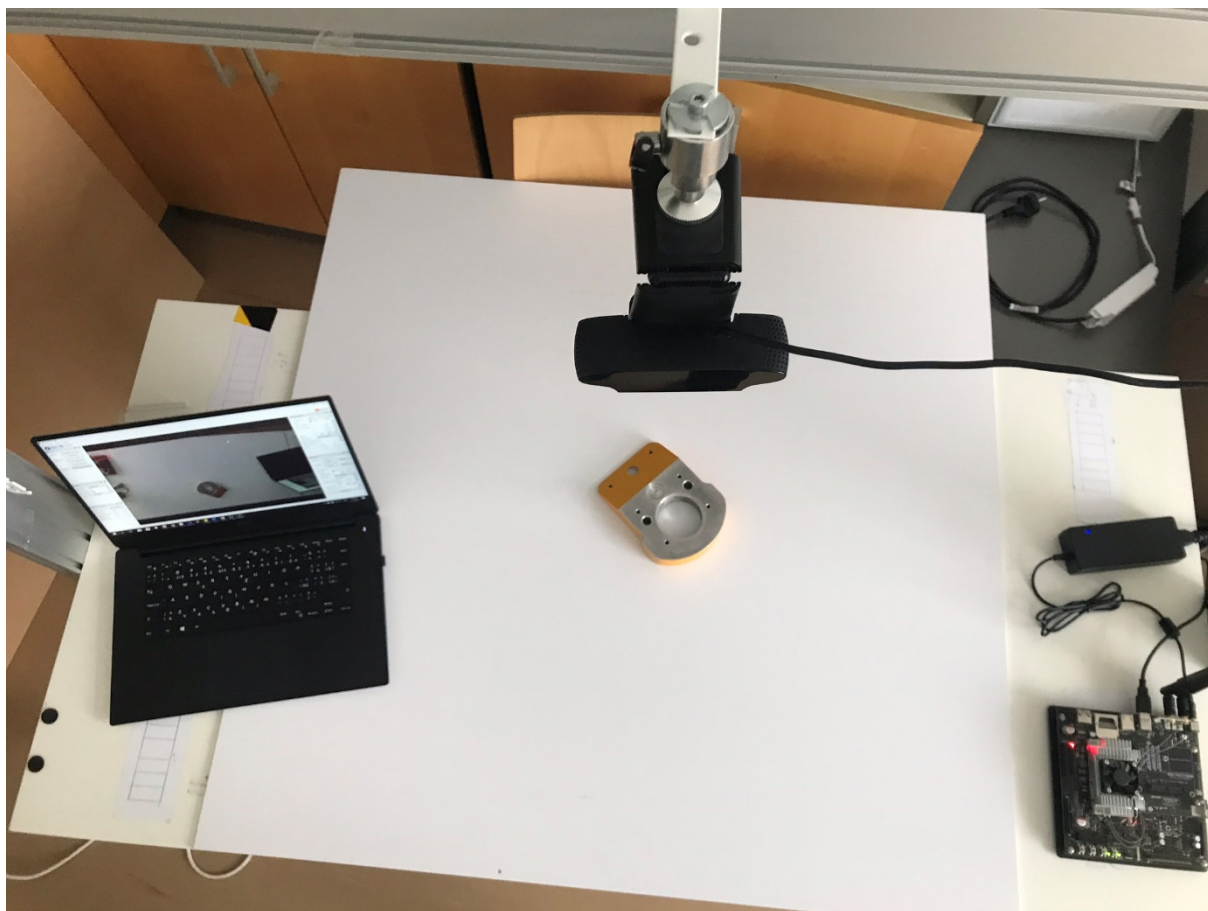
Fotografia celej zostavy s kamerou FLIR



Fotografia celej zostavy s kamerou Logitech



Fotografie zostavy pri meraní



Príloha C – Obsah priloženého CD



CD Návrh vstavaného systému inteligentného videnia na platforme NVIDIA



Diplomová práca



Fotodokumentácia



Programové vybavenie