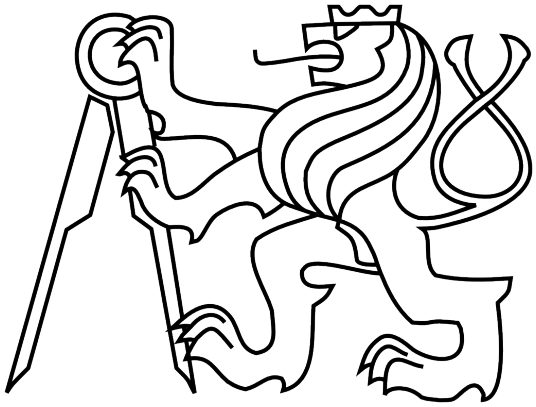
ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE

Fakulta elektrotechnická
Katedra kybernetiky



Diplomová práce

Měřicí kamera založená na Raspberry Pi

Bc. Michaela Kavková

Vedoucí práce: Doc. Ing. Jan Fischer CSc.

Studijní program: Kybernetika a robotika
Obor: Robotika

12. května 2014

České vysoké učení technické v Praze
Fakulta elektrotechnická

Katedra kybernetiky

ZADÁNÍ DIPLOMOVÉ PRÁCE

Student: Bc. Michaela K a v k o v á

Studijní program: Kybernetika a robotika (magisterský)

Obor: Robotika

Název tématu: Měřicí kamera založená na Raspberry Pi

Pokyny pro vypracování:

Navrhněte a realizujte sestavu měřicí kamery s modulem Raspberry PI. Prověřte možnost využití obrazových senzorů CMOS se sériovým i paralelním rozhraním. Při řešení připojení senzorů s paralelním rozhraním využijte výsledky prací v laboratoři videometrie katedry měření i konzultace vedoucího práce. Měřicí kamera bude zpracovávat obraz kontrastních objektů a vyhodnocovat jejich plochu, polohu, tvar, počet a další parametry. Vytvořte potřebné programy a ověřte funkci kamery na demonstrační aplikaci sledování objektů pohybujících se nezávisle v omezeném prostoru.

Seznam odborné literatury:

[1] Hlaváč, V., Sedláček,.M.: Zpracování signálu a obrazu. Skriptum, ČVUT- FEL, Praha, 2002.
[2] Heijden, F.: Image Based Measurement Systems: Object Recognition and Parameter Estimation. ISBN 10: 0471950629, Wiley, 1995.
[3] Fischer, J.: Optoelektronické senzory a videometrie. Skriptum, ČVUT, FEL, Praha, 2002.

Vedoucí diplomové práce: doc. Ing. Jan Fischer, CSc.

Platnost zadání: do konce letního semestru 2014/2015

L.S.

doc. Dr. Ing. Jan Kybic
vedoucí katedry

prof. Ing. Pavel Ripka, CSc.
děkan

V Praze dne 6. 12. 2013

Čestné prohlášení

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

V Praze dne . . . . . .

. . . . . .
Podpis autora práce

Poděkování

Chtěla bych poděkovat vedoucímu práce doc. Ing. Janu Fischerovi, CSc. za zapůjčení Raspberry Pi a jeho příslušenství a za vstřícnost při vedení, oponentovi Ing. Ottovi Havlemu, CSc., MBA za poskytnutí důležitých testovacích dat a konzultace k problematice. Dále chci poděkovat Ing. Vojtěchu Francovi, Ph.D. za rady z oblasti rozpoznávání obrazu. Nakonec chci poděkovat svým přátelům, spolužákům a blízké rodině za jejich trpělivost a toleranci při tvorbě této práce.

Abstrakt

Tato práce se zabývá zpracováním obrazu na Raspberry Pi. Cílem práce je použití modulu Raspberry Pi s obrazovými senzory CMOS se sériovým i paralelním rozhraním za účelem realizace měřicí kamery a vytvoření demonstrační aplikace zabývající se zpracováním obrazu. K dispozici je rozšiřující kamera k Raspberry Pi se sériovým rozhraním a CMOS kamera s paralelním rozhraním připojená k USB řadiči Cypress vyvinutá na katedře měření. Měřicí kamera zpracovává obraz kontrastních objektů pohybujících se v omezeném prostoru. Vyhodnocují se vlastnosti objektů jako je velikost, poloha, orientace ve 2D a zpracovává se jejich trajektorie. Práce navrhuje a implementuje demonstrační aplikaci sledování pohybu ryb v nádrži, sloužící k detekci znečištění nebo otravy vody ve vodárnách. V této práci je navržen funkční univerzální algoritmus pro detekci pohybujících se objektů ve statickém obraze, který je možný využít i v jiné oblasti. Implementace algoritmu je navržena v jazyku C++ s použitím knihovny pro zpracování obrazu OpenCV. V závěru práce jsou zhodnoceny výhody a nevýhody použití Raspberry Pi jako měřicí kamery.

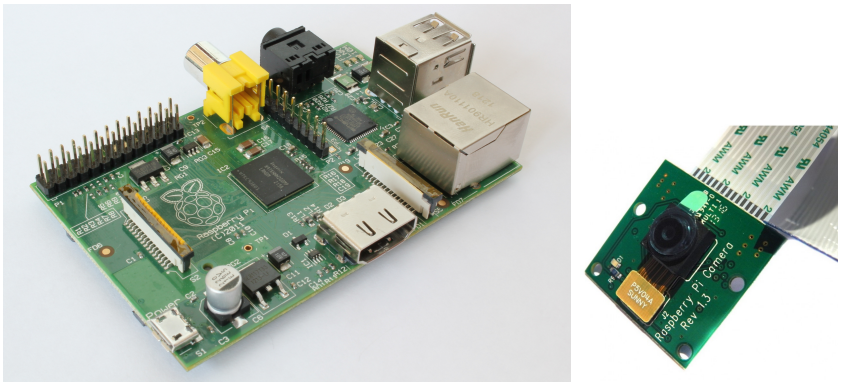
Abstract

This diploma thesis deals with the image processing using Raspberry Pi module. Goal of a work is a usage of Raspberry Pi module with connected CMOS camera sensor with serial and parallel interface in order to realize a measuring camera and creation of a demo application dealing with image processing. Raspberry Pi Camera Board with serial interface is used such as parallel interface CMOS camera attached with Cypress USB controller developed on department of measurement is used. The measuring camera processes images of contrasting objects moving in restricted area. Features and characteristics as size, position, orientation and trajectory are handled. The thesis designs and implements illustrative application based on tracking of moving fishes in a tank proposed to detect contaminated or poisoned water employed in waterworks. The functional and universal algorithm is developed in this thesis. The algorithm serves to detect moving objects in static scene which can be used in other domain. Implementation of algorithm is written in C++ programming language applying the computer vision library OpenCV. Advantages and disadvantages of Raspberry Pi usage is discussed at the end of the thesis.

Obsah

Kapitola 1

Úvod



Obrázek 1.1: Modul Raspberry Pi a kamera
Zdroj: en.wikipedia.org, www.modmypi.com

V současné době se zpracování obrazu využívá v široké škále odvětví od výzkumné oblasti mobilní robotiky až k průmyslové oblasti pro kontrolu jakosti výrobků. Stále více aplikací si vyžaduje různorodý hardware s různým výpočetním výkonem a paměťovou výbavou. Můžeme jmenovat FPGA programovatelná hradlová pole umožnující běh jednodušších aplikací s nižší paměťovou náročností, mikrokontroléry umožňující složitější výpočty nad snímaným obrazem a poskytující základní periferní obvody pro přenos dat, až k stolním počítačům s vysokým výkonem, které se používají pro off-line zpracování obrazu (například ke generování 3D scény).

Zaměřme se na použití mikrokontrolérů. Nejpoužívanějším typem mikrokontroléru je 8051, který je dodnes využíván pro paměťově i výpočetně nenáročné aplikace. V poslední době je však moderní používat k nejrůznějším účelům ARM procesory. Jejich výhodou je nízká spotřeba a vysoký výkon oproti 8051, proto se nejčastěji setkáme s tímto typem procesoru u mobilního telefonu nebo tabletu. Vývoj se dále posouvá k použití jako náhrada za klasický procesor architektur osobních počítačů. Takovým případem je modul Raspberry Pi, který je postaven na architektuře ARM11 podporující operační systémy Linuxových

distribucí.

Vznik tohoto modulu není náhodný a neměl za cíl zaplnit prázdné místo na trhu. Původní myšlenka byla rozšířit výuku programování a zpřístupnit výkonný hardware mladým studentům [1]. Hlavní podmínkou tedy bylo vyvinout výkonný hardware za dostupnou cenu. Raspberry Pi nabízí připojení obrazových snímačů více způsoby a proto byl vybrán pro tuto diplomovou práci, aby se ověřilo jak je modul vhodný pro účely videometrie. Je potřeba prozkoumat jeho možnosti použití a tomu se budu v této práci věnovat. Budu se zabývat i tím, jak zařízení uvést do provozu a jaké aplikace budou potřeba při práci s tímto zařízením. Pro ukázku správnosti by bylo vhodné navrhnout malé aplikace věnující se základním rozpoznávacím úlohám.

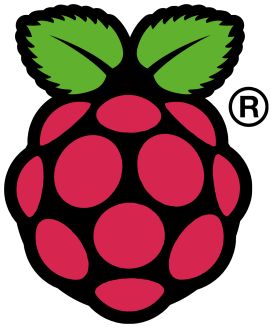
Pro použití modulu jako měřicí kamery je zapotřebí vyzkoušet více kamer s různým rozhraním. V této práci se budu zabývat dvěma typy kamer. První kamera Raspberry Pi Camera Board je dodávána výrobcem a připojuje se přes sériové rozhraní rovnou do konektoru na desce plošných spojů. Druhá kamera byla celá vyvinuta na katedře měření. Vycházím z poznatků bakalářských a diplomových prací, které se věnují vývoji hardware a návrhu firmware pro měřicí kameru. Ing. Martin Zoubek se ve své bakalářské a diplomové práci [13] věnuje vývoji desky plošných spojů s CMOS senzorem s paralelním přenosem dat. Pokračuje připojením USB řadiče Cypress CY7C68013A a přenosu dat do Windows ve své diplomové práci. Matěj Ondrička se v bakalářské práci [17] věnuje připojení tohoto USB řadiče pro Linux a své pokusy provádí také na Raspberry Pi. Očekává se užší spolupráce s panem Ondričkou při tvorbě našich prací.

Abych se dostala k důvodu, proč použít Raspberry Pi jako měřicí kameru, vraťme se k rozpoznávání obrazu. Je běžné používat měřicí kamery v průmyslových podnicích pro zajištění kvality výrobků nebo jako bezkontaktní senzory. Kamerami se dají nahradit kontrolní stanoviště, která zajišťují například vyhodnocování čistoty lahví v pivovarech nebo kontrolu kvality vody ve vodárnách. Takto automatizované pracoviště většinou vytvoří méně chyb než pracovník, který po několika minutách ztrácí pozornost. Proto zajímavým námětem pro použití měřicí kamery s Raspberry Pi je detekce a sledování ryb v nádrži pro vyhodnocování kvality vody založené na jejich pohybu a aktivitě. Při vývoji aplikace bude potřeba se nejprve seznámit s chováním ryb za normálních a abnormálních podmínek včetně možností jejich detekce. Tyto informace budu čerpat od firmy FCC Průmyslové systémy, která řešila podobný problém.

Protože bude zapotřebí data z kamer různě zpracovávat a zobrazovat, použiji v této práci knihovnu OpenCV (Open Computer Vision). Ta je velmi používanou knihovnou pro zpracování obrazu. Nabízí bohatou sbírku algoritmů pro počítačové vidění včetně příkladů a tutoriálů. Knihovna nabízí funkce v jazycích Python, C a C++. Jako programovací jazyk byl zvolen jazyk C++ kvůli doporučovaným formátům v OpenCV, které jsou objektově orientované.

Kapitola 2

Modul Raspberry Pi model B



Obrázek 2.1: Logo Raspberry Pi
Zdroj: www.raspberrypi.org

2.1 Vznik a vývoj modulu

Nápad vytvořit levný minipočítač se zrodil v roce 2005 na univerzitě v Cambridge v University of Cambridge's Computer Laboratory [2]. Tehdejší pracovník univerzity Eben Upton si všiml, že klesá počet talentovaných studentů přicházejících na školu [1]. Chtěl, aby se mladší studenti seznámili blíže s počítačem, kde vidí, jak co funguje, a naučí se pracovat se vstupně-výstupními zařízeními.

V roce 2006 započal kariéru ve firmě Broadcom, ale stále spolupracoval s univerzitou na vývoji levného počítače pro studenty. Jeho cenová hranice nesměla překročit 25 $. Nakonec vývoj skončil u čipu Broadcom 2835 postavený na architektuře ARM 11, který poskytoval i HDMI, 3D akceleraci a další periferie.

V roce 2009 byl dokončen vývoj a proto bylo nasnadě založit organizaci Raspberry Pi Foundation. Do roku 2011 se podařilo vytvořit základní desku plošných spojů, ale očekávala se jen výroba o menším objemu. Nečekaný zájem však posunul Raspberry Pi Foundation mnohem dál než očekávala a již má na svém kontě přes 2 miliony prodaných kusů [3].

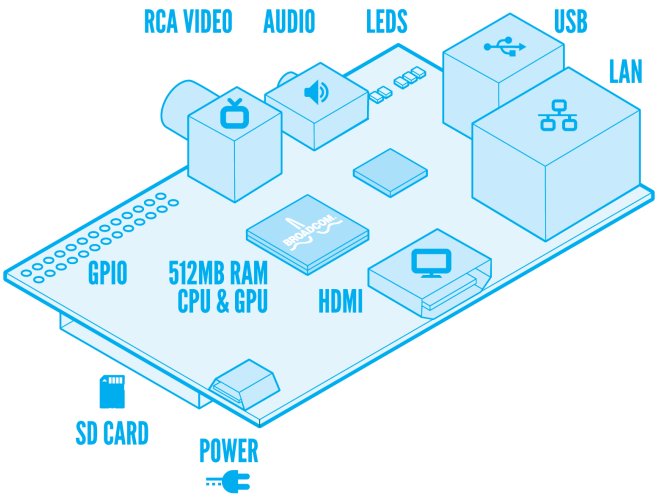
Raspberry Pi nevyužívají jen děti k výuce programování, ale existují i takoví nadšenci,

kteří ovládají starou mikrovlnou troubu přes iPhone a skenují si čárové kódy z obalů, podle nichž mikrovlnka připravuje jídlo [1]. Další neuvěřitelný nápad jak využít Raspberry Pi - nafotit zemský povrch z balónové sondy [4].

2.2 Základní parametry Raspberry Pi

Raspberry Pi (dále jen RPi) pohání procesor Broadcom BCM2835 s taktem 700 MHz typu ARM1176JZ-F s nízkým příkonem 3.5 W [5]. RPi se prodává k datu 2.5.2014 ve dvou modelech. Model A disponuje 512 MB sdílené paměti s grafickým jádrem, model B má pouze 256 MB. Oba modely se dají připojit díky HDMI a RCA výstupu k monitoru nebo televizi, zvuk se přenáší přes 3.5 mm jack. Operační systém a data jsou uložena na SD kartě. Model B nabízí 2 × USB2.0 porty s RJ45 konektorem pro LAN, která jsou připojena jako hub. Model A má pouze jedno USB2.0 a je přímo připojeno na BCM2835.

RPi má také na své desce 26 vstupně-výstupních pinů, která poskytují UART, SPI, I2C. Další piny slouží pro PWM, 3V3, 5V napájení apod. Na desce je ještě umístěn CSI (Camera Serial Interface) konektor pro připojení Raspberry Pi Camera Board, a konektor DSI (Display Serial Interface), který ale není softwarově podporován. Rozměry RPi jsou pak jen 85.6 mm × 56 mm × 21 mm. Cena modelu A je stanovena na 25$, model B stojí 35$. Na obrázku 2.2 je naznačeno umístění výše zmíněných konektorů.



Obrázek 2.2: Periferie Raspberry Pi, model B
Zdroj: www.raspberrypi.org

Existuje několik operačních systémů optimalizovaných pro RPi. Jejich výčet je následující:

- Raspbian - Založen na distribuci Debian, doporučený a nejpoužívanější OS pro RPi
- Pidora - Založen na distribuci Fedora
- RISC OS - Operační systém navržený pouze pro ARM, není to distribuce Linux ani nemá souvislost s Windows

- RaspBMC - Založen na distribuci Debian, která podporuje XBMC multimediální centrum
- Arch - Založen na distribuci Arch Linux
- OpenELEC - Linuxová distribuce s podporou XBMC multimediálního centra

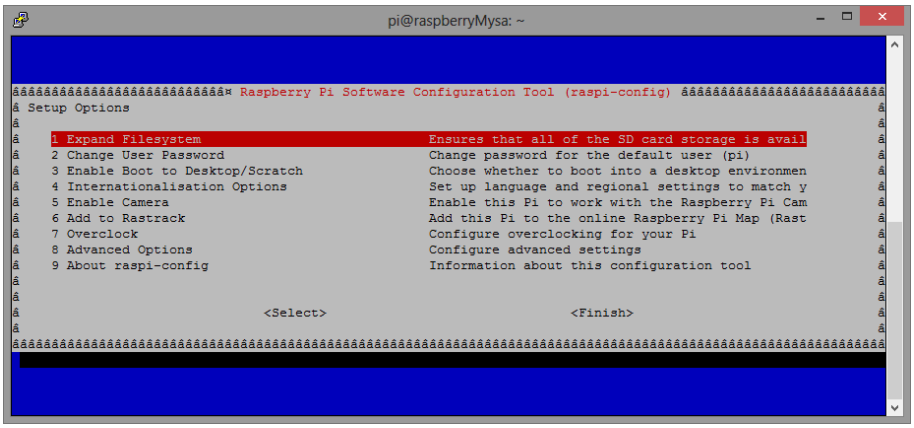
Pro naše účely byla jako nejvhodnější distribuce zvolena Raspbian. Raspbian podporuje jak konzolový výpis tak i grafické prostředí. V následujících kapitolách popíšu instalaci OS, přizpůsobení k práci na RPi bez monitoru a řešení problémů, které se vyskytly při práci.

2.3 Instalace Raspbian

Při přípravě instalace jsem využívala návodů z neoficiálních zdrojů na RPi wiki [6]. Operační systém je vždy uložen na SD kartě. Minimální kapacita by měla být 2 GB, ale musím podotknout, že karta slouží také jako úložiště veškerých dat, proto je doporučována karta s vyšší kapacitou. Použila jsem kartu 32 GB, která by měla být dostačující pro většinu aplikací. Kartu bylo potřeba nejdříve správně naformátovat a nahrát operační systém (pro Windows použit program Win32DiskImager). Karta s operačním systémem Raspbian není na Windows kompletně čitelná kvůli rozdílným formátům dat, avšak konfigurační soubory lze upravovat. Ty slouží při bootování ke správnému zavedení systému. Užitečná nastavení mohou být například:

- `config.txt`
  - proměnná `gpu_mem` přiřazuje grafickému jádru zadanou velikost paměti. Minimální hodnota je 16 MB, maximální je 448 MB.
  - proměnná `disable_camera_led` vypíná LED na kameře při pořizování videa či fotek při nastavené 1.
  - proměnná `disable_l2cache` zakazuje přístup GPU do L2 cache při nastavené 1. Výchozí hodnota je 0.
  - proměnná `sdtv_mode` nastavuje RCA výstup (0 - NTSC, 2 - PAL)
  - proměnná `sdtv_aspect` nastavuje poměr stran RCA výstupu (1 - 4:3, 2 - 14:9, 3 - 16:9)
- `cmdline.txt`
  - proměnná `ip` přiřazuje statickou IP adresu, např. 169.254.0.3

Pro první spuštění doporučuji připojit Raspberry k monitoru podporující HDMI, použít klávesnici a myš, a připojit Raspberry do internetové sítě. Raspberry nemá žádné zapínací ani vypínací tlačítko, pouze se připojí napájecí kabel Micro-USB do běžné USB nabíječky (popř. notebooku) poskytující stejnosměrné napětí 5 V a proud minimálně 750 mA. Operační systém automaticky začne bootovat a po prvním spuštění nabízí pouze konzolový výpis.



Obrázek 2.3: Nabídka `raspi-config`

Příkazem `raspi-config` se vyvolá nastavení operačního systému. Raspbian automaticky nerozšiřuje použitelnou velikost SD karty do plné velikosti, proto je dobré spustit první příkaz `Expand Filesystem`. Také je dobré změnit si heslo, jinak by se do našeho Raspberry mohl dostat cizí uživatel zvenčí (každé Raspberry má z výroby jméno `pi` a heslo `raspberry`).

Pokud chceme používat grafické uživatelské rozhraní, můžeme ho zapnout třetím příkazem z Obrázku 2.3. Čtvrtý příkaz slouží ke změně jazyka, které jsem ale ponechala v základním nastavení na angličtinu. Pro práci s kamerou je třeba ji systémově zapnout příkazem `Enable Camera`. Nastavení umožňuje přetaktování čipu i jednotlivých bloků SOC, které jsem ale nezkoušela. Frekvence je potřeba ještě nastavovat v konfiguračním souboru.

Ostatní položky nabídky již nejsou podstatné pro naše účely. Po restartu Raspberry máme k dispozici funkční minipočítač (Obrázek 2.4). Pokud je Raspberry připojeno k internetu, můžeme provést aktualizaci příkazy

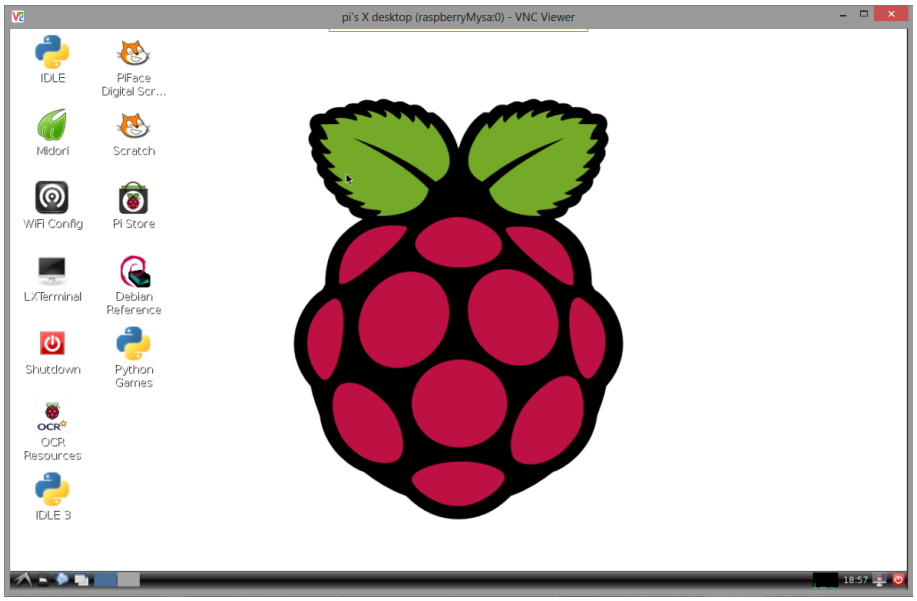
```
> sudo apt-get update
> sudo apt-get upgrade
```

Důležitým negativním poznatkem je, že RPi nemá hodiny a aktuální čas získává z internetu. Po odpojení ze sítě odpočítává čas od tohoto odpojení.

Pokud nemáme k dispozici monitor, klávesnici a myš, můžeme Raspberry Pi používat i bez nich, pomocí protokolů vzdálených služeb SSH, VNC, SFTP a další, jejichž použitím se věnuji v následujících sekcích.

2.4 Použití Internetu na RPi

Připojení k ethernetové síti a zvláště Internetu je v dnešní době základním požadavkem pro jakýkoliv počítač. Ne jinak je tomu i u Raspberry Pi. Protože nemá integrovanou baterii pro napájení hodin, získává aktuální čas z Internetu. Dalším důvodem proč RPi připojit k Internetu je instalace a aktualizace programů a knihoven. Buď můžeme RPi připojit



Obrázek 2.4: Plocha operačního systému

přes LAN kabel do RJ45 konektoru umístěného vedle USB hubu, nebo můžeme použít kompatibilní WiFi adaptér do USB.

Při použití vzdálených přístupů jako jsou SSH, VNC nebo SFTP je vždy nejdůležitější znát IP adresu Raspberry Pi. Pokud se RPi a náš počítač nachází v jedné síti, jejíž router nám přiděluje IP adresy z DHCP serveru a k němuž máme přístup, nepíšeme statickou adresu do konfiguračního souboru `cmdline.txt`. Rezervaci statické adresy nastavíme v routeru pro naše RPi zařízení. Pokud jsme však spojeni sítí, ve které nefiguruje DHCP server (například přímé spojení PC $\leftrightarrow$ RPi kabelem), musíme IP adresu Raspberry nejprve přidělit přes konfigurační soubor s tím vědomím, aby IP adresa našeho počítače měla stejnou masku.

2.5 Protokol SSH na RPi

Protokol SSH (Secure Shell) slouží k zabezpečenému přístupu přes příkazovou řádku do jiného počítače. Tento protokol můžeme použít v programu PuTTY, kde zadáme pouze IP adresu. Po zadání přihlašovacích údajů se už nacházíme v příkazové řádce vzdáleného počítače, v našem případě Raspberry Pi. Na RPi je protokol SSH ve výchozím nastavení zapnutý. V nabídce `raspi-config` lze SSH vypnout příkazem `Advanced Options`.

Při spouštění programů vyžadujících grafické zobrazení si musíme uvědomit, že program vždy vrátí chybu zobrazení. V PuTTY lze použít také protokol X11 pro zobrazení desktopového rozhraní, ale myslím si, že vhodnější řešení je použít protokol VNC.

2.6 Protokol VNC na RPi

Protokol VNC (Virtual Network Computing) slouží výhradně k přenosu grafického prostředí po ethernetu. Je ideálním prostředníkem pro použití RPi bez monitoru, klávesnice a myši. K přenosu nám stačí VNC Server pro Raspberry a VNC Viewer pro náš počítač s Windows. Server pro RPi nainstalujeme příkazem

```
> sudo apt-get install tightvncserver
```

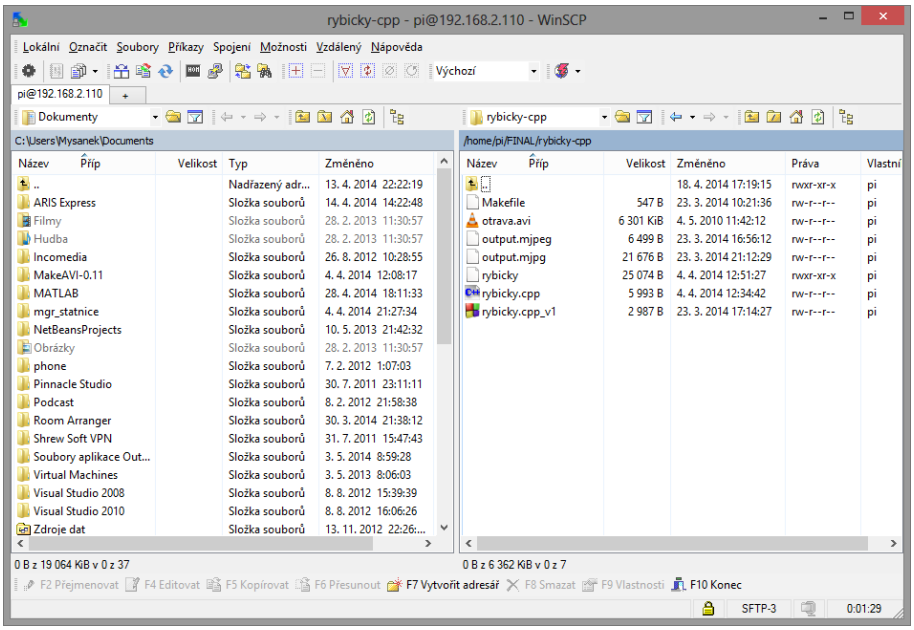
a dále používáme jen příkaz

```
> vncserver :0 -geometry 1920x1080 -depth 24 -dpi 96
```

pro full HD rozlišení. Pro rychlejší spouštění jsem vytvořila bashovský skript `svnc.sh`, který můžeme spouštět po každém startu.

2.7 Přenos souborů mezi RPi a osobním počítačem

Data můžeme přenášet mezi RPi a naším počítačem více způsoby. Nabízí se protokol SFTP (Secure File Transfer Protocol) a SCP (Secure Copy) protokol. Oba tyto protokoly se kombinují v programu WinSCP určeným k zabezpečenému přenosu dat mezi počítačem s Windows a vzdáleným serverem. Grafické rozhraní připomíná Salamander nebo Windows Commander.



Obrázek 2.5: WinSCP grafické rozhraní

2.8 Instalace knihovny OpenCV

Nejrozšířenější knihovnou pro zpracování obrazu je OpenCV zejména kvůli velkému množství základních funkcí, jejichž použití je tak říkajíc uživatelsky přívětivé. Některé funkce se funkčností podobají funkcím v Matlabu, dokonce přístupy k maticím se přizpůsobují matlabovskému zápisu. Instalace knihoven OpenCV a jejich začlenění do systému je pro zkušeného linuxového uživatele jistě snadné, ale pro méně zkušené uživatele se nabízí skript, který všechnu práci provádí sám. Použila jsem skript pro instalaci verzí OpenCV 2.4.5 a OpenCV 2.4.6.1 z git repozitáře Jay Rambhia [7]. Následujícími příkazy nastavíme stažený skript jako spustitelný a poté ho spustíme.

```
> chmod +x opencv2_4_5.sh
> ./opencv2_4_5.sh
```

Obdobně spouštíme i další verze skriptů v [7]. Nezapomeňme vždy po jakékoliv instalaci provést aktualizaci a upgrade instalovaných aplikací příkazy

```
> sudo apt-get update
> sudo apt-get upgrade
```

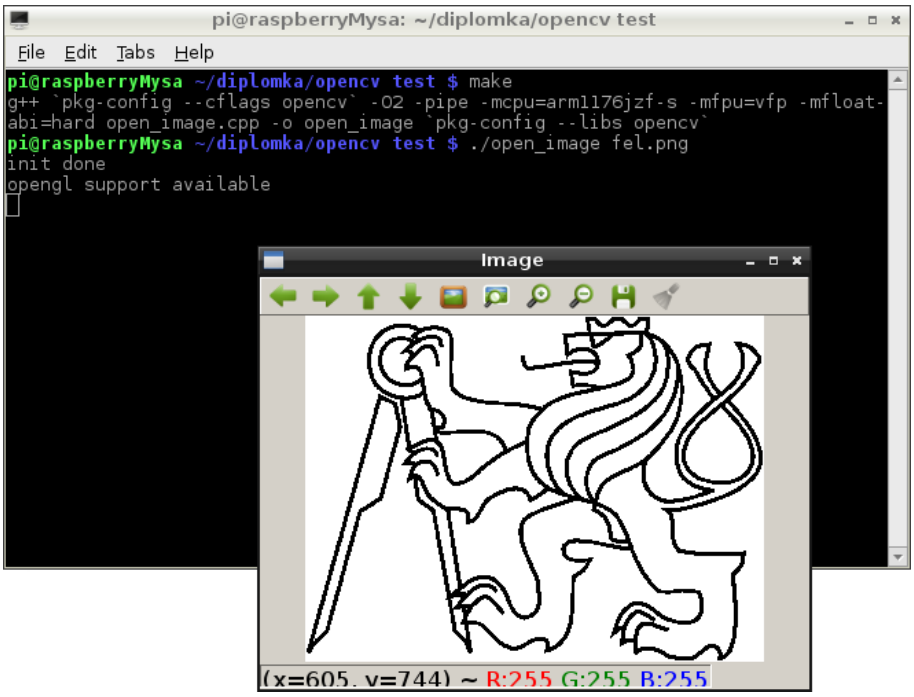
2.9 Test funkčnosti instalací

Důležité je vyzkoušet správnost instalací v sekci 2.8. Troufám si říci, že instalace OpenCV bude trvat i několik hodin oproti instalaci stejným skriptem na stolním počítači, kde trvá jen několik minut. Může se i stát, že instalace nedoběhne do konce kvůli chybějícím balíčkům, které musíme doinstalovat ručně. V neposlední řadě se nám také může stát, že nezávislou instalací odlišných balíčků se pokazí reference na OpenCV v systému a bude potřeba OpenCV reinstalovat (resp. konkrétní balík, který byl poškozen).

Knihovna nabízí testovací zdrojové kódy pro překlad, ale mě se osvědčil můj vlastní kód s vlastním Makefile, ve kterém je připsáno, kde má překladač hledat odkazy na OpenCV. V DVD příloze je ve složce `Test OpenCV` přiložen soubor `open_image.cpp` a `Makefile`. Program se přeloží a spustí následujícím způsobem.

```
> make
> ./open_image cvut.png
```

Program otevře obrázek cvut.png v novém okně a čeká, dokud ho nezavřeme klávesou Escape (Obrázek 2.6).



Obrázek 2.6: Testovací program - zobrazení obrázku

Kapitola 3

Připojení obrazových senzorů

3.1 Raspberry Pi Camera Board

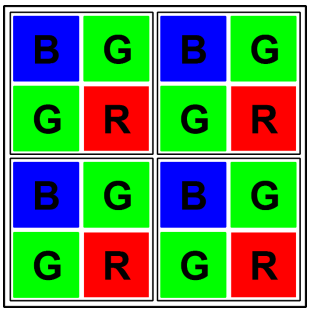
Obrázek 3.1: Raspberry Pi Camera board
Zdroj: www.embeddedcomputer.nl

Raspberry Pi Camera board je zařízení určené pouze pro Raspberry Pi. Dokáže snímat jak video, tak vytvářet jednotlivé snímky. Je určena začátečníkům a nabízí vestavěné funkce v Raspbianu pro snímání videa a obrázků.

RPi kamera se skládá z plošného spoje o rozměrech 25 mm × 24 mm s CMOS senzorem OmniVision OV5647 o velikosti 3.67 mm × 2.74 mm [8], a ohebného 15-žilového kabelu. Senzor poskytuje rozlišení 2592 × 1944 pixelů a dokáže snímat HD video o rozlišení 1920 × 1080 pixelů s frekvencí 30 FPS (Framerate Per Second, tj. počet snímků za sekundu). V tabulce 1 je uvedeno 6 dalších formátů a jejich hodnoty FPS pro použitý CMOS senzor OV5647 [9]. Senzor obsahuje 2624 sloupců a 1956 řádků, přičemž vždy 4 krajní řádky (resp. sloupce) nejsou aktivní. Obrazové body jsou uspořádány do Bayerovy mřížky. Bayerova mřížka je uspořádání barevných pixelů podle Obrázku 3.2.

Formát	Rozlišení	FPS	Metoda škálování
5 Mpx	2592×1944	15	plné rozlišení
1080p	1920×1080	30	ořez
960p	1280×960	45	ořez, převzorkování/slučování
720p	1280×720	60	ořez, převzorkování/slučování
VGA	640×480	90	ořez, převzorkování/slučování
QVGA	320×240	120	ořez, převzorkování/slučování

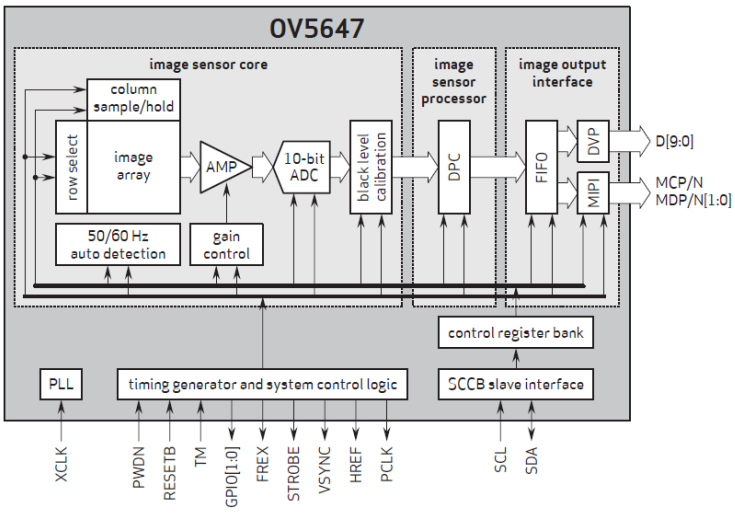
Tabulka 3.1: Tabulka formátů a FPS pro senzor OV5647



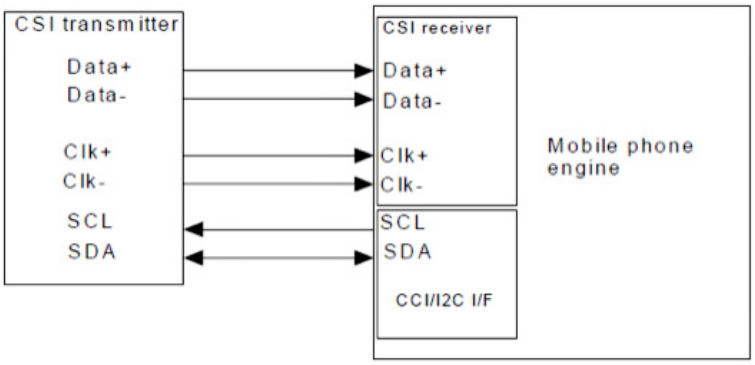
Obrázek 3.2: Uspořádání pixelů do Bayerovy mřížky

3.1.1 Rozhraní RPi kamery

Na Obrázku 3.3 [8] je blokové schéma senzoru. Zaměřme se na rozhraní MIPI v pravé části schématu.



Obrázek 3.3: Blokové schéma CMOS senzoru OV5647



Obrázek 3.4: Blokové schéma CSI rozhraní
Zdroj: www.mipi.org/specifications/camera-interface

RPi kamera se připojuje do CSI konektoru na RPi desce. CSI (Camera Serial Interface) je součástí MIPI (Mobile Industry Processor Interface) rozhraní, jež je určeno pro komunikaci s komponentami v mobilních zařízeních [10]. CSI je sériové rozhraní sloužící k přenosu dat většinou mezi digitální kamerou a mobilním zařízením. Mobilním zařízením je myšleno software a hardware zajišťující telekomunikační funkce nebo aplikační úlohy. V našem případě je přijímačem Raspberry Pi. Rozhraní poskytuje datovou a řídicí část. Datová část je pouze jednosměrná z kamery do RPi s hodinovým signálem. Řídicí část je obousměrná a kompatibilní s I2C rozhraním.

Obrázek 3.5 ilustruje Raspberry Pi s připojenou RPi kamerou v praktickém plastovém obalu, jehož dírka sloužící k otevírání krabičky je využita jako otvor pro kameru.

Obrázek 3.5: RPi s připojenou kamerou v praktickém obalu

3.1.2 Instalace driveru pro RPi kameru

Data z kamery můžeme získávat programy `raspivid` a `raspistill`, ale nemůžeme je využívat v rámci zdrojového kódu vlastního programu, proto jsem využila existující Video4Linux2 drivery pro RPi kameru. Ty připojí kameru jako video zařízení, které je viditelné v `/dev/video0`.

Na začátku pokusů nebyl k dispozici oficiální ovladač, proto jsem použila UV4L driver ze serveru Linux Projects [11]. Pro instalaci sledujeme návod v [11], ale je důležité po instalaci provést příkaz

```
> sudo rpi-update
```

bez něhož není možné správně kameru "otevřít". Po každém spuštění RPi je třeba ovladač nejdříve zapnout příkazem

```
> uv4l --driver raspicam --auto-video_nr --encoding yuv420 --width 1920 --height 1080
```

který připojí kameru na `/dev/video0` o rozlišení 1920×1080 ve formátu YUV420. Je možné spouštět více ovladačů zároveň na pozice následující `/dev/video1`, atd. Dalším příkazem lze měnit parametry na již připojeném zařízení

```
> v4l2-ctl --set-fmt-video=width=640,height=480,pixelformat="H264" -d /dev/video0
```

V prosinci 2013 byl vydán oficiální driver, který se nainstaluje a spustí následovně.

```
> rpi-update
> sudo modprobe bcm2835-v4l2
```

Při potížích je třeba postupovat podle návodu v RPi fóru [12]. Použití oficiálního driveru považuji za vhodnější řešení, protože nikdy nedošlo k nedostupnosti kamery. Zapíná se jednodušeji a nabízí i přívětivější nastavení. Pro nahrávání videa, snímání obrázků, zobrazení podporovaných formátů a aktuální nastavení slouží příkazy

```
> v4l2-ctl --set-fmt-video=width=1920,height=1088,pixelformat=4
> v4l2-ctl --stream-mmap=3 --stream-count=100 --stream-to=video.h264

> v4l2-ctl --set-fmt-video=width=2592,height=1944,pixelformat=3
> v4l2-ctl --stream-mmap=3 --stream-count=1 --stream-to=image.jpg

> v4l2-ctl --list-formats

> v4l2-ctl -V
```

3.1.3 Test funkčnosti RPi kamery

Pro test funkčnosti zařízení jsem vytvořila jednoduchý program `captureImage`, který otevře video zařízení umístěné v `/dev/video0` a ve smyčce čte data z kamery. Uvnitř programu je přepínač `displayWindows`, kterým můžeme vypnout zobrazování a tím zvýšit počet snímků za sekundu. Do konzole se zapisují hodnoty FPS. Překlad a spuštění provedeme příkazy

```
> make
> ./captureImage
```

Výpis z konzole při vypnutém zobrazování okna (`displayWindows = false`) pak vypadá například

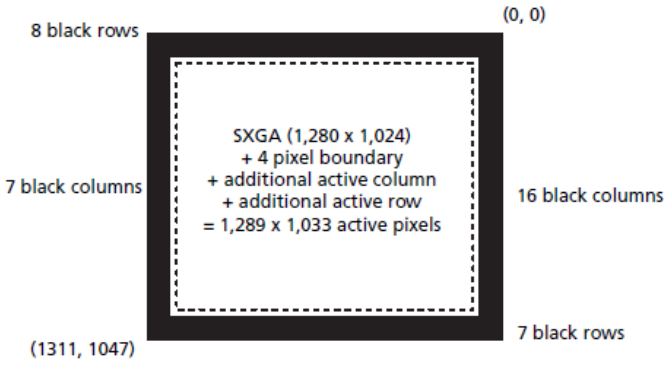
```
counter: 140, 35.00 fps
counter: 141, 35.25 fps
counter: 142, 35.50 fps
counter: 143, 35.75 fps
counter: 144, 36.00 fps
counter: 145, 36.25 fps
counter: 146, 36.50 fps
counter: 147, 36.75 fps
counter: 148, 37.00 fps
counter: 149, 37.25 fps
counter: 150, 30.00 fps
```

3.2 Kamera s USB řadičem Cypress

Takto nazvaná kamera sestává ze dvou desek plošných spojů, z nichž senzorová deska je navržena Ing. Janem Šedivým, a řídicí deska Ing. Martinem Zoubkem [13]. Na kameru je připevněn kroužek pro upevnění objektivu Tevidon Zeiss 25mm f/1.4. Kamera s objektivem je zobrazena na Obrázku 3.6.

Obrázek 3.6: Kamera s USB řadičem Cypress s objektivem Tevidon Zeiss 25mm f/1.4

Hlavní částí senzorové desky je obrazový CMOS senzor s názvem MT9M001 od firmy Micron. Tento senzor je monochromatický poskytující rozlišení 1280×1024 pixelů s rychlostí 30 snímků za sekundu. Senzor obsahuje 1312 sloupců a 1048 řádků, z nichž je několik krajních sloupců, resp. řádků, neaktivních, jak je naznačeno na Obrázku 3.7.



Obrázek 3.7: Uspořádání pixelů senzoru Micron MT9M001 [14]

Řídicí deska je osazena rychlým USB řadičem Cypress CY7C68013A s jádrem 8051. Podporuje standard USB2.0 s přenosovou rychlostí až 480 Mb/s, má 16 kB vyrovnávací paměti, a 8/16 bitovou sběrnici FIFO [15]. Podle měření pana Ondričky nejvyšší dosažená přenosová rychlost zaručující bezeztrátový přenos mezi RPi a řadičem je 11,4 Mb/s [17].

3.2.1 Použití kamery s aplikací `BulkLoopApp`

Protože pro řadič Cypress CY7C68013A není vydán žádný oficiální driver pro Linux, pan Ondrička naprogramoval vlastní ovladač pomocí knihovny `libusb`. Není to ovladač v pravém slova smyslu, který by běžel na pozadí a připojil by kameru jako video zařízení. Aplikace s názvem `BulkLoopApp` pouze zprostředkovává vyžádání dat k přenosu, jejich samotný přenos z FIFO paměti řadiče, a uspořádání do vhodného formátu.

Pro použití této aplikace nejprve musíme nainstalovat knihovny `libusb`

```
> sudo apt-get install libusb-1.0-0
> sudo apt-get install libusb-1.0-0-dev
```

Poté můžeme aplikaci přeložit a spustit. Program funguje správně spuštěním jen jako správce (root). Spouštíme ho tedy se slovem `sudo`.

```
> make
> sudo ./BulkLookApp
```

Program se nás dotáže, jaký typ zařízení chceme po USB obsluhovat. Vybereme sběrnici a zařízení s ID výrobce `0x04b4` a ID produktu `0x1004`. V další nabídce máme tři možnosti, jak má aplikace pokračovat.

1. zobrazit informace o zařízení
2. spustit vyčítání dat z kamery
3. ukončit program

Výběrem druhé možnosti začnou data proudit do programu, ve kterém se ukládají do struktury v OpenCV. Od toho momentu se dají sejmutá data zobrazit jako obrázek v náhledu, nebo použít k dalšímu zpracování.

3.2.2 Test funkčnosti kamery s USB řadičem

Pro test této kamery s Raspberry Pi využijeme práce pana Ondričky, aplikaci `BulkLoopApp`. Do této aplikace vložíme stejný testovací kód jako v případě testování RPi kamery. Ve smyčce se získávají data, jež se v případě `displayWindows = true` zobrazí v okně náhledu. Do konzole pak proudí jen informace o hodnotě FPS. K překladu pak použijeme `Makefile` a program spustíme stejným způsobem jako v předchozí sekci 3.2.1. Protože je aplikace stejná jako v případě RPi kamery, má i stejný výstup, proto už ho zde nebudu zmiňovat.

3.3 Srovnání obou kamer

Na obou typech kamer a způsobu přenosu jsem spustila stejný program zmíněný v sekcích 3.1.3 a 3.2.2. Program jsem zkoušela se zapnutým i vypnutým náhledem snímaného obrazu. Při testech nebylo RPi připojené ethernetovým kabelem k Internetu, protože mu ubírá výkon. Výpočet spočíval v měření celkového času t_c pro zpracování $n = 5000$ snímků, podělený tímto počtem.

$$\text{FPS} = \frac{t_c}{n} \tag{3.1}$$

Průměrné naměřené hodnoty snímků za sekundu jsem shrnula do tabulky 3.2. Získané hodnoty platí pro rozlišení obrázku 640×480 pixelů.

Je vidět, že jako nejlepší řešení se nabízí RPi Camera Board a to zejména díky rychlému přenosu po CSI rozhraní. USB je na RPi všeobecně pomalejší, proto výsledek kamery s USB řadičem je očekávaný.

Pro porovnání ještě uvedu tabulku, jak si kamera s řadičem Cypress vede v Debianu 7.5 Wheezy běžícím na notebooku Lenovo s procesorem Intel Core-i7 2.2 GHz s 8 GB RAM a grafickou kartou NVIDIA GeForce GT740 2GB. Porovnáme-li tabulky 3.3 a 3.2, je zřejmé že USB přenos dat z kamery s řadičem Cypress již nelze více zrychlit použitím výkonnějšího hardwaru, protože výkonný notebook nemá ztráty při zobrazování náhledu a hodnoty FPS jsou vlastně totožné.

Kamera	FPS bez náhledu	FPS s náhledem
RPi Camera Board	30.13	21.74
USB kamera	12.95	12.89

Tabulka 3.2: Průměrné hodnoty FPS s programem `captureImage`/`BulkLoopApp` na RPi

Kamera	FPS bez náhledu	FPS s náhledem
USB kamera	13.18	13.03

Tabulka 3.3: Průměrné hodnoty FPS s programem `BulkLoopApp` na notebooku

Kapitola 4

Demonstrační aplikace

Na ukázku práce Raspberry Pi jako měřicí kamery jsem vyvinula tři aplikace. První se zabývá detekcí laserové stopy v obraze. Druhá detekuje kontrastní objekty v obraze. Obě aplikace využívají k detekci prahování. Třetí aplikace již využívá k detekci metodu zvanou *background subtraction*. Všechny tři aplikace využívají funkce z knihovny OpenCV, kterou si rozebereme blíže. V následujících sekcích budu čerpat především z dokumentace knihovny OpenCV verze 2.4.9.0 v [18].

4.1 Práce s obrazovými daty v OpenCV

Obrázky v OpenCV 2.4.9.0 v jazyku C++ se ukládají do třídy s názvem `Mat`. Objekt této třídy může být n-dimenzionální matice s jedním nebo více kanály. Hodnoty buněk mohou nabývat celých čísel, plovoucích čísel i imaginárních čísel, a mohou mít různý rozsah. Na ukázku předvedu kód, který vytvoří matici `image` obsahující obraz o velikosti 640×480 pixelů v 8bit stupních šedi naplněných hodnotou 128.

```
Mat image(Size(640, 480), CV_8UC1, 128);
```

Podobných konstruktorů existuje celá řada a nabízí opravdu rozmanité inicializace. Předchůdci objektů `Mat` byli `CvMat`, `IplImage` a ještě starší `CvMatND`, se kterými můžeme dodnes pracovat, ale doporučuje se hlavně `Mat` formát. Oproti `IplImage` nemusíme na objekt `Mat` volat dealokační funkci `release()` při každé změně ukazatele, objekt si ho zavolá sám.

Při práci s maticemi nabízí objekt `Mat` klasické operace, které známe například z Matlabu. Někdy je dokonce zápis velmi podobný. Jako ukázku nabízím používané funkce v implementovaných aplikacích v tabulce 4.1.
Přístup k jednotlivým buňkám matice se děje pomocí operátoru `at`, který pochází z typu <`vector`>. Následující příkaz vybere prvek na 10. řádku a 10. sloupci. Tímto způsobem přístupu je zajištěno, že se nejdříve ověří ukazatel na buňku, a poté se do ní přistupuje.

```
image.at<float>(9, 9);
```

Operace	Příkaz v Matlabu	Zápis v OpenCV
součet	`C = A + B`	`add(A, B, C, noArray(), -1);`
rozdíl	`C = A - B`	`subtract(A, B, C, noArray(), -1);`
násobení skalárem	`A = B * x`	`A = B * x;`
nulová matice	`A = zeros(y,x);`	`A = Mat::zeros(Size(x,y), CV_8UC1);`
jednotková matice	`A = eye(x);`	`A = Mat::eye(Size(x,x), CV_8UC1);`
prahování	`A(B>t) = 256; A(B<=t) = 0;`	`A = B > t;`
výřez	`A = B(1:y, 1:x);`	`A = B(Rect(1, 1, x, y));`

Tabulka 4.1: Zápisy operací v Matlabu a OpenCV

4.1.1 Převody mezi prostory barev

Pro práci s kamerami na RPi budeme potřebovat jen převod z RGB (resp. BGR) do stupňů šedi. Převod realizujeme funkcemi

```
cvtColor(frame, frame, CV_RGB2GRAY, 1);
cvtColor(frame, frame, CV_BGR2GRAY, 1);
```

Všimněme si, že na tříkanálový obrázek `frame` přiřazujeme jednokanálový, aniž bychom dealokovali paměť. To je přesně ten případ, kdy nemusíme volat dealokační příkaz `release()`. Rozdíl mezi dvěma funkcemi jsou v řazení kanálů R-G-B nebo B-G-R. Pro převod uvažujeme rovnici

$$Y = 0.299R + 0.587G + 0.114B \tag{4.1}$$

Pro rozdělení barevných kanálů používáme funkci `split`

```
Mat RGB[3];
split(frame, RGB);
```

4.1.2 Zápis a čtení videí, obrázků

Abychom mohli vyčítat snímky z kamer připojených do `/dev/videoN`, musíme zařízení zachytit do proměnné `VideoCapture`. Po ujištění, že je spojení otevřené, je zaručeno, že data budeme moci číst. Následující ukázka kódu naznačuje správný postup k získání jednoho snímku

```
VideoCapture capture(0);
if(!capture.isOpened()) {
    cout << "Capture failure" << endl;
    return -1;
}
```

```
if (!capture.read(frame)) {
    cout << "Unable to read frame. Returning." << endl;
    return -1;
}
capture.release();
```

Funkce `capture.read(frame)` zavolá `grab` (sejme holá data), dekóduje je a uloží aktuální snímek do `frame`. Podobně čteme i snímky z videa uloženého na disku.

```
VideoCapture capture("video.avi");
```

V testovacím programu pro OpenCV instalaci jsem ještě využívala načítaní obrázku z disku příkazem

```
Mat image;
image = imread("cvut.png", CV_LOAD_IMAGE_COLOR);
```

Pro ukládání videa se používá třída `VideoWriter`.

```
VideoWriter outputVideo;
int FPS = 25;
outputVideo.open("output.avi", CV_FOURCC('M','J','P','G'), FPS, ...
    ... frame.size(), true);
if (!outputVideo.isOpened()) {
    cout << "ERROR: Can't write to video.avi!" << endl;
    return -1;
}
outputVideo.write(frame);
```

a pro ukládání obrázků

```
imwrite("outputImage.jpg", frame);
```

4.1.3 Náhledy

K zobrazení snímku můžeme využít náhledu v okně v OpenCV. V následujícím kódu vytvoříme nové okno s názvem `Image` a zobrazíme v něm logo ČVUT. Nesmíme zapomenout okno zase zničit.

```
Mat frame = imread("cvut.png", CV_LOAD_IMAGE_COLOR);
namedWindow("Image", CV_WINDOW_NORMAL);
imshow("Image", frame);
waitkey(0);
destroyWindow("Image");
```

4.1.4 Kreslení obrazců

OpenCV má implementované funkce pro kreslení úseček, kružnic, elips, obdélníků a textů do obrázku. Uvedeme si příklady použití

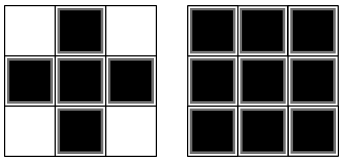
```
// červená úsečka tloušťky 2
line(frame, Point(x1, y1), Point(x2, y2), Scalar(0, 0, 255), 2, 8);
// zelená kružnice
circle(frame, Point(x, y), r, Scalar(0, 255, 0), 2, 8);
// červená elipsa s osami a, b a úhlem phi
ellipse(frame, Point(x, y), Size(a, b), phi, 0, 360, Scalar(0, 0, 255), 2, 8, 0);
// modrý obdélník
rectangle(frame, Rect(x, y, width, height), Scalar(255, 0, 0), 2, 8, 0);
// zelený text
putText(frame, "Hello world!", Point(x, y), 0, 0.5, Scalar(0, 255, 0), 2, 8, false);
```

4.2 Použitá teorie v aplikacích

4.2.1 Morfologické operace

Ve třetí aplikaci používám matematické morfologie, které pomáhají předzpracovat a segmentovat obraz [19, 20]. Představíme si pouze dvě binární morfologie *dilatace* a *eroze*.

Morfologické operace se provádějí mezi dvěma bodovými množinami, kde jedna je většinou menší a říkáme jí *strukturní element* [19]. V mém případě jsem použila element velikosti 3×3 s motivem kříže (Obrázek 4.1a).



(a) (b)

Obrázek 4.1: Strukturní elementy

Morfologická operace *dilatace* se zapisuje rovnicí 4.2, kde E^2 je Euklidovský dvourozměrný prostor, A a B jsou bodové množiny. Dilatace sčítá každý bod množiny A s každým bodem množiny B. Použitím elementu z obrázku 4.1b objekt narůstá.

$$X \oplus B = \{p \in E^2 : p = x + b, x \in X, b \in B\} \tag{4.2}$$

Morfologická operace *eroze* je duální k *dilataci* a zapisuje se rovnicí 4.3. Při použití strukturního elementu z Obrázku 4.1b objekty opticky ztrácí okraje a tím dojde k rozdělení objektů na menší.

$$X \ominus B = \{p \in E^2 : p + b \in X \text{ pro každé } b \in B\} \tag{4.3}$$

4.2.2 Momenty

Momenty nazýváme statistické parametry pravděpodobnostního rozdělení s náhodnou veličinou. Náhodná veličina je v tomto případě jasová hodnota obrazových bodů a momenty vyjadřují charakteristiku jejího pravděpodobnostního rozdělení v obraze [20]. Uvažujme hodnotu pixelů jako binární. Uvedeme si momenty, které budeme používat v této práci.

Obecný moment $(p+q)$-tého řádu dvourozměrné veličiny $f(x,y)$ se počítá rovnicí 4.4. Moment m_{00} nám vyjadřuje plochu objektu. Těžiště objektu pak spočítáme pomocí rovnice 4.5.

$$m_{pg} = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} x^p\, y^q\, f(x,y)\,\mathrm{d}x\,\mathrm{d}y \tag{4.4}$$

$$x_c = \frac{m_{10}}{m_{00}}\,, \qquad y_c = \frac{m_{01}}{m_{00}} \tag{4.5}$$

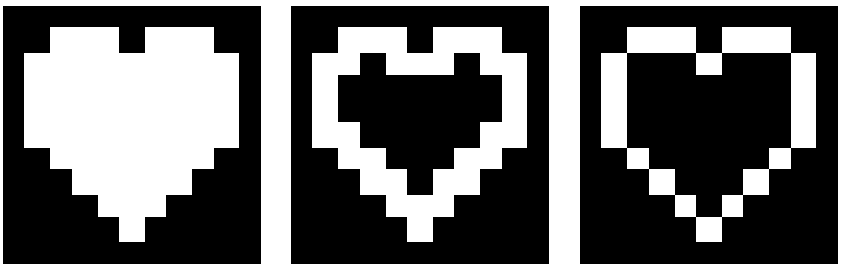
Díky *centrálním momentům* pak můžeme spočítat směr nejdelší osy obdélníku ohraničujícího objekt. Centrální moment se počítá podle rovnice 4.6 a orientace úhlu φ pak rovnicí 4.7.

$$\mu_{pg} = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} (x-x_c)^p\,(y-y_c)^q\, f(x,y)\,\mathrm{d}x\,\mathrm{d}y \tag{4.6}$$

$$\varphi = \frac{1}{2}\arctan\left(\frac{2\mu_{11}}{\mu_{20}-\mu_{02}}\right) \tag{4.7}$$

4.2.3 Reprezentace tvaru segmentovaných objektů

Hranice objektů se nazývají kontury. Uvažujeme-li 4-okolí, mají hraniční body mezi sebou vzdálenost 1. Uvažujeme-li 8-okolí, mají body mezi sebou vzdálenost 1 nebo $\sqrt{2}$. Kontury získáme jako rozdíl originálu (Obrázek 4.2a) a erodovaného obrazu. Použitím strukturního elementu z Obrázku 4.1a dostaneme kontury jako 8-okolí (Obrázek 4.2c). Použijeme-li zaplněný strukturní element z Obrázku 4.1b, dostaneme konturu jako 4-okolí (Obrázek 4.2b).



(a) Originál (b) Kontura 4-okolí (c) Kontura 8-okolí

Obrázek 4.2: Příklad kontury

Obrazové body kontury se pak sdružují do řetězců tak, aby každý bod měl jednoho předchůdce a jednoho následovníka. Zápis do řetězce je možný několika způsoby zmíněných

v [20], pro naše účely je ale vhodný zápis do řetězce, který obsahuje souřadnice jednotlivých bodů, tak jak jdou za sebou. Příklad z Obrázku 4.2c pak zapisujeme jako

```
<[2,3], [2,4], [2,5],  [3,6], [2,7], [2,8], [2,9], [3,10], [4,10], [5,10], [6,10],
 [7,9], [8,8], [9,7], [10,6], [9,5], [8,4], [7,3],  [6,2],  [5,2],  [4,2],  [3,2]>
```

Takový řetězec můžeme zjednodušit v místech, kde se vyskytují vodorovné, svislé nebo diagonální úsečky. Potom je zápis kratší.

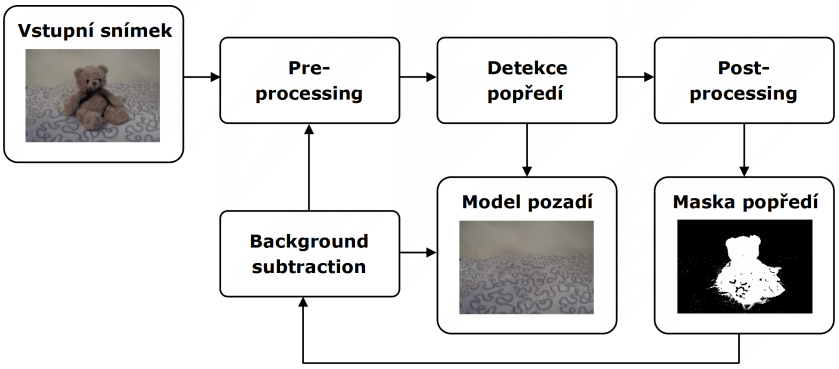
```
<[2,3], [2,5], [3,6], [2,7], [2,9], [3,10], [6,10], [10,6], [6,2], [3,2]>
```

4.2.4 Modelování pozadí a popředí statického obrazu

Modelování pozadí (*background subtraction*) je běžně používaný způsob, jak segmentovat objekty v popředí. Využívá se například v monitorování dopravy nebo k detekci pohybujících se lidí v obraze. Existuje několik způsobů, jak pozadí získat a já zde zmíním jen dva způsoby, které jsem používala pro své experimenty s detekcí ryb.

Oba zvolené způsoby jsou rekurzivní, to znamená, že v paměti máme jeden model pozadí $B_t(x, y)$, který je při každém novém snímku $I_t(x, y)$ přepočítán [21]. To má za důsledek nízkou paměťovou náročnost, kvůli které je *background subtraction* používanou metodou detekce pohybu v reálném čase.

Na Obrázku 4.3 je naznačeno blokové schéma obecného algoritmu detekce pozadí a popředí. Pre-processingem je myšleno například odstranění šumu Gaussovským filtrem. Další bloky schématu si představíme v následujících odstavcích.



Obrázek 4.3: Blokové schéma obecného algoritmu *background subtraction*

Uvažujeme-li snímky ve stupních šedi, mají proměnné $B_t(x, y)$, $I_t(x, y)$ jeden kanál. Pokud pracujeme s barevnými snímky, mají kanály tři, a pro každý kanál přepočítáváme pozadí zvlášť.

Nyní představím první způsob výpočtu modelu pozadí. Využívá Gaussovského rozdělení pro nahrazení jasové složky obrazového bodu. Každý obrazový bod modelu pozadí je reprezentován parametry Gaussovského rozdělení (μ, σ). Model pozadí v čase 0 se inicializuje

podle rovnic 4.8 a 4.9 z prvního snímku $I_0(x, y)$. Parametr σ_0 se nastaví na určenou počáteční hodnotu rozptylu.

$$\mu_0(x, y) = I_0(x, y) \tag{4.8}$$

$$\sigma_0^2(x, y) = \sigma_{\text{default}}^2 \tag{4.9}$$

Tyto parametry se aktualizují v každém novém snímku pomocí klouzavého průměru. Klouzavý průměr parametrů (μ, σ) pro obrazový bod (x, y) se počítá podle rovnic 4.10 a 4.11 [23]. Parametr α určuje míru zapomínání klouzavého průměru, tedy jak rychle zapomene na snímek v čase t. Obvykle se používá hodnota $\alpha \in \langle 0.001, 0.1 \rangle$.

$$\mu_t(x, y) = \alpha I_t(x, y) + (1 - \alpha)\mu_{t-1}(x, y) \tag{4.10}$$

$$\sigma_t^2(x, y) = \alpha \left[I_t(x, y) - \mu_t(x, y) \right]^2 + (1 - \alpha)\sigma_{t-1}^2 \tag{4.11}$$

Obrazové body popředí jsou potom body (x, y) vyhovující rovnici 4.12, kde k je práh obvykle volen jako $k = 2.5$ (tj. bod je v popředí, pokud rozdíl od střední hodnoty je větší než 2.5-násobek rozptylu) [23]. Vhodné hodnoty parametrů α a k byly zkoumány v článku [22], odkud také pocházejí hodnoty uvedené v této práci.

$$\frac{|I_t(x, y) - \mu_t(x, y)|}{\sigma_t(x, y)} > k \tag{4.12}$$

Reprezentace hodnoty obrazového bodu pozadí se neomezuje jen na použití Gaussovského rozdělení, ale používá se i směs Gaussovských rozdělení s neměnným i adaptivním počtem směsí. K výpočtu parametrů směsí Gaussiánů se používá algoritmus *Expectation Maximization* (EM).

EM algoritmus maximalizuje věrohodnost $\Pr(\mathrm{X}|\theta)$, kde X je množina pozorovaných vzorků (například dvourozměrných bodů), a θ reprezentuje parametry směsi Gaussiánů. Algoritmus zavádí skrytou proměnnou Q, která určuje příslušnost vzorků do shluků. Výpočet se provádí ve dvou krocích. První krok *expectation* odhadne rozdělení skryté proměnné Q pro dané vzorky a parametry θ směsi Gaussovských rozdělení. Druhý krok *maximization* upravuje hodnoty θ tak, aby byla maximalizována věrohodnost pro dané Q [24].

Druhý způsob modelování pozadí, který v této práci představím, nevyužívá statistického modelu, ale používá k reprezentaci modelu jasovou hodnotu obrazových bodů. Pro inicializaci pozadí $B_0(x, y)$ použijeme první snímek $I_0(x, y)$ (rovnice 4.13). Potom s každým novým snímkem aktualizujeme $B_t(x, y)$ pomocí klouzavého průměru pro každý obrazový bod (rovnice 4.14).

$$B_0(x, y) = I_0(x, y) \tag{4.13}$$

$$B_t(x, y) = \alpha I_t(x, y) + (1 - \alpha)B_{t-1}(x, y) \tag{4.14}$$

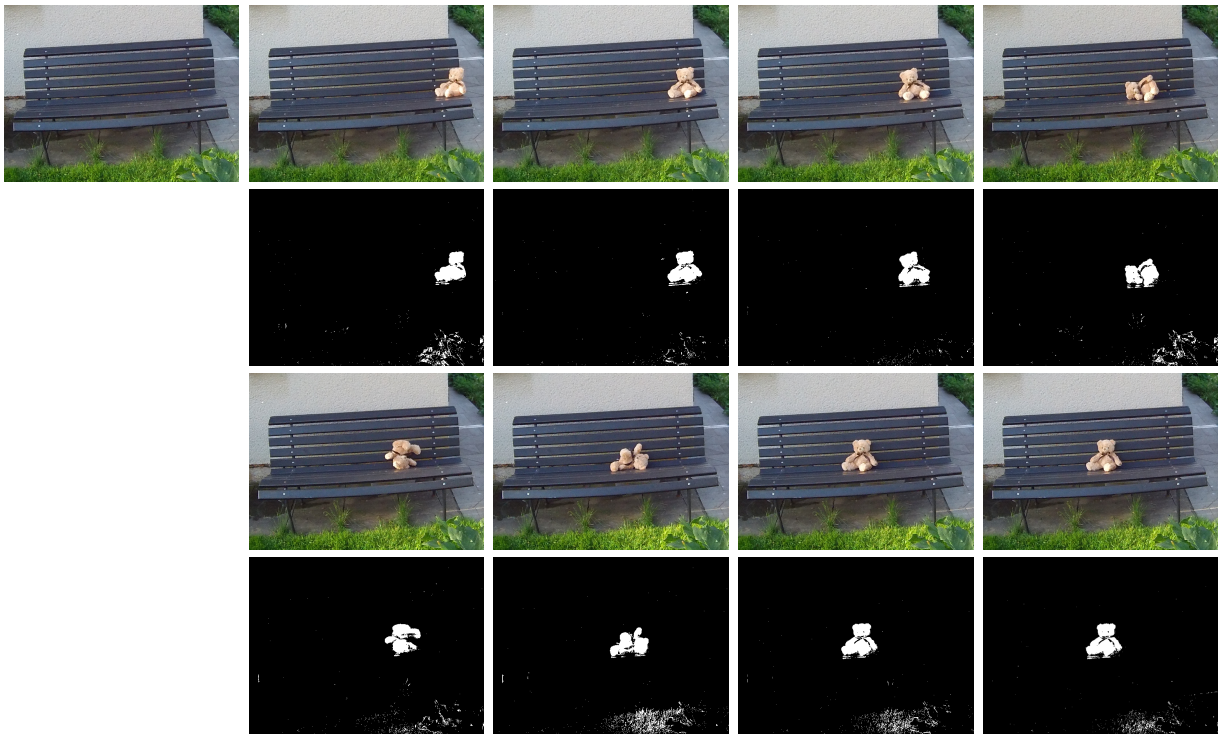
Objekt v popředí je pak složen z obrazových bodů vyhovující rovnici 4.15. Parametr k se volí pro všechny obrazové body stejný, což může být jediná nevýhoda tohoto přístupu oproti modelování pomocí Gaussiánů, kde tento práh závisí na rozptylu jednotlivých obrazových bodů zvlášť.

$$|B_t(x, y) - I_t(x, y)| > k \tag{4.15}$$

Pro obě metody používáme post-processing, který spočívá v další úpravě popředí. Nejčastěji používáme morfologické operace (*dilatace, eroze, otevření, uzavření*). V dalším kroku je možné odstranit menší objekty (nazveme je *bloby*), které nejsou validní pro segmentaci, např. pohyb listů stromů ve větru, který zanechá v popředí stopy větší než můžeme odstranit morfologickými operacemi.

V neposlední řadě se nám v popředí mohou objevit tzv. duchové (*ghosts*). Ty vzniknou, pokud se pohybující objekt zastaví na místě a chvíli setrvá. Jeho stopa je zanesena do pozadí a po opětovném rozpohybování nám generuje ducha, i když na jeho místě už žádný objekt není. To lze odstranit tzv. *optical flow* testem, který vyjme takové *bloby*, jejichž pozice se dlouho nemění. Tato metoda má i svá negativa, protože jako *ghost* se může detekovat i validní objekt, který se jen na chvíli zastavil. Výpočet optického toku (*optical flow*) zde nezmiňuji a je možné ho nastudovat v [20].

Jako ukázka segmentace metodou *background subtraction* je na Obrázku 4.4 model pozadí a sekvence snímků. Výpočtem popsaným rovnicí 4.15 pro $k = 35$ dostaneme binarizovaný model popředí bez post-processingu, který je zobrazen ve druhé řadě na Obrázku 4.4.



Obrázek 4.4: Získané popředí ze sekvence snímků

4.2.5 Prokládání objektu elipsou

Máme-li množinu dvourozměrných bodů, kterou chceme proložit elipsou, použijeme pro výpočet hodnoty obecných momentů m_{00}, m_{10}, m_{01} a centrálních momentů μ_{11}, μ_{20}, μ_{02}.

Z nich sestavíme kovarianční matici velikosti 2×2, která má na diagonále hodnoty rozptylu σ^2 a mimo diagonálu leží kovariance $cov(x, y)$ (rovnice 4.16).

$$M = \begin{pmatrix} \sigma_x^2 & cov(x,y) \\ cov(x,y) & \sigma_y^2 \end{pmatrix} = \frac{1}{m_{00}} \begin{pmatrix} \mu_{20} & \mu_{11} \\ \mu_{11} & \mu_{02} \end{pmatrix} \tag{4.16}$$

Velikost os elipsy a, b spočteme jako odmocniny vlastních čísel kovarianční matice M (rovnice 4.17, 4.18).

$$a = \sqrt{\lambda_1} \tag{4.17}$$

$$b = \sqrt{\lambda_2} \tag{4.18}$$

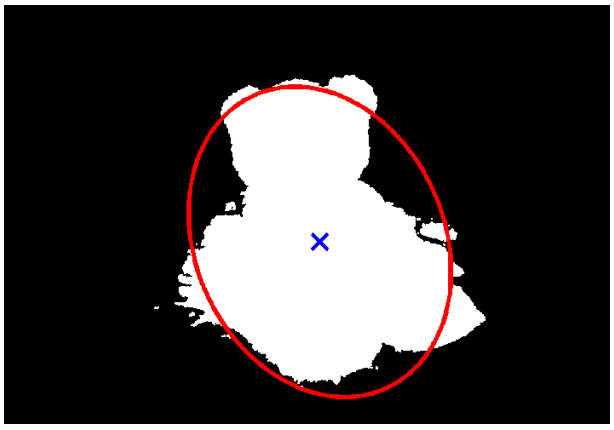
Směr osy a spočteme podle rovnice 4.19.

$$\varphi = \frac{1}{2} \arctan \left(\frac{2\mu_{11}}{\mu_{20} - \mu_{02}} \right) \tag{4.19}$$

Nakonec spočteme střed elipsy vztahem 4.20.

$$x_0 = \frac{m_{10}}{m_{00}}, \qquad y_0 = \frac{m_{01}}{m_{00}} \tag{4.20}$$

Proložíme-li takovým způsobem objekt z diagramu na Obrázku 4.3, dostaneme jeho aproximaci elipsou (Obrázek 4.5).



Obrázek 4.5: Objekt proložený elipsou

4.3 Segmentace I - Laser

Motivací k této úloze je sledování stopy laseru, například pomocí otáčecího mechanismu. Laser je promítán na rovný povrch a našim úkolem je ho v obraze najít a trackovat. Data přenášíme z kamery pomocí OpenCV. Očekáváme jeden objekt vysokého jasu s malou plochou. Výstupem programu je poloha objektu a jeho velikost.

4.3.1 Rozbor řešení úlohy Laser

Máme k dispozici červený laser, proto můžeme použít pouze červenou složku obrazu. Stopa laseru se dá snadno detekovat prahováním daného červeného kanálu. Uvažujeme-li pouze jediný objekt bez odlesků, nemusíme používat adaptivní prahování. Pro tuto demonstrační úlohu není třeba hledat optimální práh, ale zvolíme ho experimentálně. V následném binárním obraze budeme hledat střed a plochu oblasti, kde se nachází laser. K tomu využijeme momenty popsané v sekci 4.2.2. Pro monochromatickou kameru samozřejmě neuvažujeme barevné kanály a prahujeme jediný kanál šedé škály.
Následné vykreslení trajektorie je realizováno spojením dvou bodů - současného a předchozího. Nevykreslujeme žádné body, pokud nedetekujeme ve dvou po sobě jdoucích snímcích nějaký objekt.

Klademe důraz na použití co nejméně alokované paměti, protože na RPi je paměť a práce s velkým množstvím dat časově drahá.

4.3.2 Implementace úlohy Laser

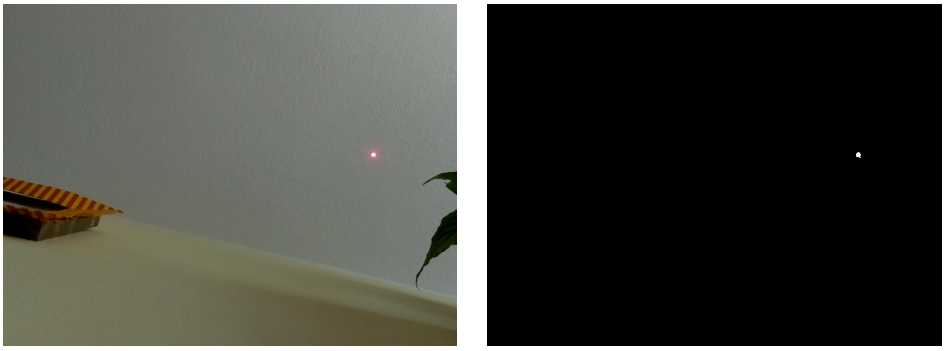
Implementace programu je velmi krátká a slouží jako ukázka použití OpenCV na Raspberry Pi. Popisuji zde implementaci pro RPi kameru. Pro USB kameru bude jen rozdíl v práci s barevnými kanály. Stopu laseru reprezentujeme třídou `Laser`. V paměti uchováváme aktuální (`current`) a předchozí (`last`) nalezený objekt.

```
struct Laser {
    int area;
    float x;
    float y;
};
```

Začneme inicializací kamery, která je popsaná v sekci 4.1.2, a pokračujeme smyčkou, která trvá, dokud uživatel nezmáčkne Escape. Ve smyčce je vždy načten obrázek a rozdělen na jednotlivé kanály. Musíme si vždy uvědomit, jaké pořadí kanálů snímáme. V našem případě snímáme RGB, ale OpenCV vykresluje v pořadí BGR. Proto pořadí změníme a uložíme obrázek do `frame2`.

```
int from_to[] = { 0,2, 1,1, 2,0 };
Mat frame2(frame.size(), CV_8UC3);
mixChannels(&frame, 1, &frame2, 1, from_to, 3);
Mat BGR[3];
split(frame, BGR);
```

Červený kanál je oprahován funkcí zmíněnou v 4.1 s prahem o hodnotě 200. Tato hodnota byla experimentálně určena jako robustní pro tuto jednoduchou aplikaci. Výsledek prahování je vidět na Obrázcích 4.6a a 4.6b.



(a) Originál (b) Binární obrázek

Obrázek 4.6: Příklad prahování snímku s laserovou stopou

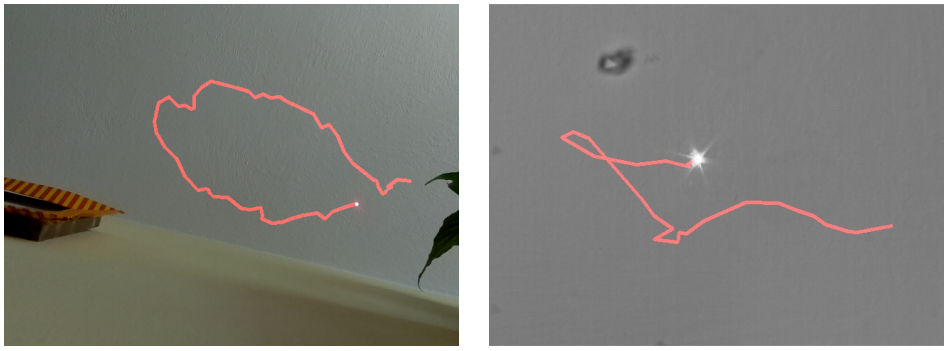
Při následném určení polohy a plochy stopy laseru nám postačí výpočet momentů ze sekce 4.2.2. Získané hodnoty zapisujeme do logu s názvem `log.txt` spolu s časovou známkou a hodnotou aktuálního FPS.

```
Moments moments;
moments = cv::moments(BGR[2] > 200, true);
current.area = moments.m00;
if (current.area !=0) {
    current.x = moments.m10/current.area;
    current.y = moments.m01/current.area;
    log<<t<<"\t"<<fps;
    printLog(log, current);
} else {
    current.x = -1;
    current.y = -1;
}
```

Pro vykreslování stopy do obrázku platí následující pravidla. Pokud předchozí i současný objekt má plochu větší než 0, tedy oba byly nalezeny, vykreslíme úsečku. Nakonec je současný objekt přiřazen jako předchozí a smyčka se vrací na začátek. Následující kód implementuje nastíněný postup.

```
if (last.area != 0 && last.x >= 0 && current.area != 0) {
    line(draw, Point(last.x, last.y), Point(current.x, current.y), ...
        ... Scalar(0, 0, 255), 2, 8);
}
add(frame2, draw, frame2, noArray(), -1);
if (displayWindows) imshow("Image", frame2);
last = current;
```

Výsledná trajektorie pak může vypadat jako na Obrázku 4.7.



(a) RPi kamera (b) USB kamera

Obrázek 4.7: Příklad nalezené trajektorie stopy laseru na obou typech kamer

Data zapsaná do logu pak mají následující formát.

Timestamp	FPS	Center x	Center y	Area
6730000	10.4286	502.96	195	52
6820000	10.571	490.23	194.15	79
6910000	10.714	478.68	193.35	84
7000000	10.857	463.59	194.61	83
7100000	11	447	197.27	75
7190000	11.143	440.33	194.62	84
7290000	11.286	439.29	188.69	65
7380000	11.429	439.96	176.26	89
7470000	11.571	448.67	164.94	103

4.3.3 Návod na spuštění úlohy Laser

Složka s názvem `Laser RPi` obsahuje program pro RPi kameru. Druhá složka `Laser USB` je určena pro kameru s USB řadičem. Obě složky se nacházejí v DVD příloze A. Dále budu popisovat jen program pro RPi kameru. Složka tedy obsahuje

- `captureLaser.cpp`
- `Makefile`

Uvnitř kódu v `captureLaser.cpp` si můžeme všimnout proměnné `displayWindows`, která při hodnotě `false` nezobrazuje náhled snímků. Tím je aplikace mnohem rychlejší. Následující tabulka shrnuje časovou náročnost. Všimněme si, že ač se zdála RPi kamera rychlejší, tak převody RGB do stupňů šedi ji nakonec zpomalují. Při zapnutém i vypnutém náhledu obě kamery zvládají problém stejně rychle.

displayWindows	FPS s RPi kamerou	FPS s USB kamerou
false	10.409	10.364
true	6.375	6.365

Tabulka 4.2: Rychlost zpracování detekce laserové stopy

Překlad provádíme pomocí připraveného `Makefile` a spuštění provádíme následovně.

```
> make
> ./captureLaser
```

Záznamy o objektech se zapisují do souboru `log.txt`. Pokud tento soubor existuje, jeho obsah je vymazán a nahrazen novým.

4.4 Segmentace II - Prahování

Motivace k vytvoření následující demo aplikace bylo určování kvality a čistoty vody z chování ryb. Při tvorbě tohoto programu jsem ještě neměla k dispozici reálná videa nebo fotografie, které by zachycovaly akvárium s rybami, které se používají pro detekci toxických látek. Z toho důvodu jsem uvažovala následující myšlenku. Představme si nádrž nebo akvárium se světelným zdrojem v pozadí. Rybí těla vytvoří v obraze tmavé objekty, které lze detekovat a sledovat jejich pohyb. Protože jsem si představovala problém velmi zjednodušeně, je program postaven na segmentaci obrazu prahováním.

4.4.1 Rozbor řešení úlohy Prahování

Úkolem je nalezení kontrastních objektů na světlém pozadí. Pro zjednodušení použijeme bílý papír s vytištěnými černými obrazci, nebo tmavé objekty položené na bílé podložce. Snímané obrázky převedeme do stupňů šedi podle rovnice 4.1, protože nepotřebujeme k segmentaci barevné hodnoty.

Protože uvažujeme pouze bílé pozadí a černé objekty, kde nejsou žádné rušivé elementy, tak obraz můžeme segmentovat prahováním. Práh je volen experimentálně.

Abychom nalezli oddělené regiony a tím segmentovali jednotlivé ryby, použijeme reprezentaci kontur do řetězce (sekce 4.2.3). Následným použitím momentů pro každý nalezený objekt spočítáme těžiště a plochu nalezených ryb, podobně jako v úloze Laser.

Pro účely trackování bylo potřeba vymyslet způsob, jak přiřazovat nalezené ryby v čase t k rybám z předchozího snímku v čase $t-1$. Existuje více způsobů jak trackovat objekty, ale já přistoupila k nejjednodušší reprezentaci. Pokud se v minulém snímku objevil pouze jeden objekt a jeden jsme detekovali, přiřadí se jim stejné ID. Pokud jsme v minulém snímku nenalezli žádný objekt, přiřadí se objektům v čase t ID od 1 do n_t (velikost

množiny v čase t). Ve zbylých případech spočítáme vzdálenostní matici o velikost $n_{t-1} \times n_t$ podle vzorce 4.21.

$$D(j,i) = (x_t - x_{t-1})^2 + (y_t - y_{t-1})^2 \,, \qquad \text{pro každé } j \in (1, n_{t-1}),\ i \in (1, n_t) \tag{4.21}$$

Potom pro každý objekt z času t najdeme nejbližší objekt z času $t-1$ jako minimum ze vzdálenostní matice. Pokud je počet $n_t > n_{t-1}$, tak zbylým objektům přiřazujeme vždy o 1 vyšší ID než bylo nejvyšší přiřazené. Takto určené dvojice pak vykreslujeme do snímku jako trajektorie.

Pro zvýraznění označíme objekty v obraze ohraničujícím rámečkem zvaným *Bounding Box*. To je nejmenší možný obdélník s hranami rovnoběžnými s osami x a y, do kterého se dá vepsat nalezený objekt.

Vypočítané parametry objektů ukládáme do logovacího souboru spolu s časovou známkou a hodnotou průměrného FPS.

4.4.2 Implementace úlohy Prahování

Hledané objekty budeme reprezentovat třídou `Fish` následující struktury.

```
struct Fish {
    int id;
    int area;
    float x;
    float y;
};
```

Instance objektu obsahuje ID, které souvisí s ID objektu z předchozího snímku. Současné (`current`) a předchozí (`last`) objekty uchováváme ve vektorech.

```
vector<Fish> current;
vector<Fish> last;
```

Pro inicializaci kamery a celého programu postupujeme stejným způsobem jako v aplikaci Laser. Ve smyčce pak opakujeme následující postup. Po načtení dat do instance třídy `Mat` ve formátu RGB převedeme každý snímek do BGR pro správné vykreslení a následně do odstínů šedi pro prahování.

```
int from_to[] = { 0,2, 1,1, 2,0};
mixChannels(&frame, 1, &frame2, 1, from_to, 3);
frame2.copyTo(frame);
cvtColor(frame2, frame2, CV_BGR2GRAY, 1);
```

Pro prahování můžeme použít konstantní práh, který jsme nejprve experimentálně určili na hodnotu 70. Protože očekáváme vždy světlé pozadí a tmavé objekty, neuvažujeme adaptivní prahování ani optimální práh. Použijeme inverzní prahování, takže všechny obrazové body s vyšším jasem než 70 nastavíme na 0. Obrazové body s nižším jasem než 70 nastavíme na 255. Dostáváme obraz jako na Obrázku 4.8b.

```
threshold(frame2, frame2, 70, 255, 1);
```

Jak už jsem zmínila, z reprezentace kontur můžeme nalézt spojité oblasti, které symbolizují jednotlivé nalezené objekty. Použijeme tedy výpočet kontur funkcí `findContours`. Ta používá reprezentaci jako seznam v hierarchii, kde v horní úrovni jsou vnější kontury a v nižší úrovni jsou pak kontury děr. Kontura je aproximována do vodorovných, svislých a diagonálních úseček, jako jsem naznačila v sekci 4.2.3.

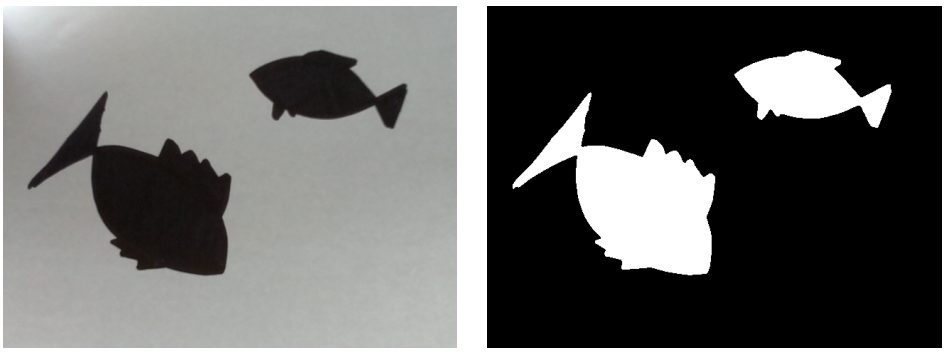
```
vector<vector<Point> > contours;
vector<Vec4i> hierarchy;
findContours(frame2, contours, hierarchy, CV_RETR_CCOMP, CV_CHAIN_APPROX_SIMPLE, ...
    ... Point(0, 0) );
```

Konturu a její vnitřek nastavíme na hodnotu 255. Proměnná `conComp` je pak hledaná spojitá komponenta v obraze, segmentovaná ryba.

```
Mat conComp;
conComp = Mat::zeros(frame.size(), CV_8U);
drawContours(conComp, contours, i, Scalar(255), CV_FILLED, 8, hierarchy, 1);
```

Pro každou rybu pak spočítáme pomocí momentů ze sekce 4.2.2 plochu a polohu. Všechny objekty s plochou větší než 500 ukládáme do vektoru `current`. Do náhledu vykreslíme ohraničující rámeček `BoundingBox` (Obrázek 4.8b).

```
Moments moments;
moments = cv::moments(conComp, true);
if (moments.m00 > 500) {
    Fish temp;
    temp.area = moments.m00;
    temp.x = moments.m10/temp.area;
    temp.y = moments.m01/temp.area;
    current.push_back(temp);
    Rect boundingBox = boundingRect(contours[i]);
    rectangle(frame, boundingBox, Scalar(0, 255, 0), 2, 8, 0);
}
```



(a) Originální obrázek (b) Objekty nalezené prahováním

Obrázek 4.8: Příklad segmentace obrazu na základě prahování

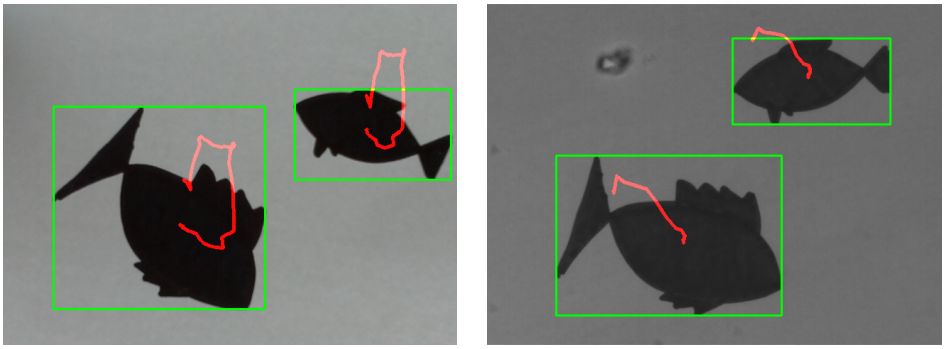
Přiřazování ID ryb je realizováno v níže přiloženém kódu způsobem popsaným v sekci 4.4.1.

```
int c = current.size();
int l = last.size();
if (c == 1 && l == 1) {
    current[0].id = last[0].id;
} else if (last.empty()) {
    for (int cur = 0; cur < c; cur++) {
        current[cur].id = cur + 1;
    }
} else {
    Mat diff(Size(c, l), CV_32FC1);
    vector<int> proc;
    int maxid = 0;
    for (int j = 0; j < l; j++) {
        for (int cur = 0; cur < c; cur++) {
            diff.at<float>(j,cur) = pow(current[cur].x - last[j].x, 2) + ...
                                    ... pow(current[cur].y - last[j].y, 2);
            if (j == 0) proc.push_back(cur);
        }
        if (maxid < last[j].id) maxid = last[j].id;
    }
    int iter;
    (c <= l) ? iter = c : iter = l;
    for (int cur = 0; cur < iter; cur++) {
        Point minLoc;
        double minVal;
        minMaxLoc(diff, &minVal, NULL, &minLoc, NULL, noArray());
        if (minVal < 1600) {
            current[minLoc.x].id = last[minLoc.y].id;
            diff.row(minLoc.y) = diff.row(minLoc.y)*FLT_MAX;
            diff.col(minLoc.x) = diff.col(minLoc.x)*FLT_MAX;
            for (int del = 0; del < proc.size(); del++) {
                if (proc[del] == minLoc.x) {
                proc.erase(proc.begin() + del);
                break;
                }
            }
        }
    }
    for (int p = 0; p < proc.size(); p++) {
        current[proc[p]].id = maxid + 1 + p;
    }
}
last = current;
current.clear();
```

Jak je možné si všimnout na konci kódu, po přidělení ID objektům uložíme do proměnné `last` hodnoty z proměnné `current` a poté `current` vymažeme. Smyčka se pak vrací na začátek a dojde ke zpracování dalších snímků, dokud uživatel nestiskne Escape.

Nezmiňuji se zde o kreslení trajektorie, která je stejná jako u aplikace Laser. To samé platí i pro logovací soubor `log.txt`, jehož zápis jsem do výše zmíněného kódu nezařazovala kvůli přehlednosti. Taktéž ukázkový kód nepopisuje program určený pro USB kameru (rozdíly jsou v práci s barevnými kanály a vykreslováním). Ukázková trajektorie je zobrazena na Obrázcích 4.9a a 4.9b. Textový výstup pak má podobu například

```
Timestamp FPS    ID Center x  Center y  Area
11880000  5.1667 1  519.34    416.59    21719
12050000  5.25   1  519.6     416.85    21611
12340000  5.3333 1  519.17    416.55    21719
12630000  5      1  519.2     416.59    21715
12870000  5.0769 1  519.36    416.55    21727
13040000  5.1538 1  519.3     416.57    21723
13040000  5.1538 2  423.31    40.058    2795
13280000  5.2308 1  519.14    416.51    21722
13280000  5.2308 2  408.76    59.762    4113
13280000  5.2308 3  36.148    217.04    3269
```



(a) RPi kamera (b) USB kamera

Obrázek 4.9: Příklady nalezených trajektorií na obou typech kamer

4.4.3 Návod na spuštění úlohy Prahování

Složka s názvem `Thresh RPi` obsahuje program pro RPi kameru. Druhá složka `Thresh USB` je určena pro kameru s USB řadičem. Obě složky jsou přítomny na DVD v příloze A. Dále budu popisovat jen program pro RPi kameru. Složka obsahuje

- `captureObject.cpp`
- `Makefile`

Uvnitř kódu je opět proměnná `displayWindows`, která slouží k zapínání a vypínání náhledu snímaného obrazu. Další volitelná proměnná je `findID`, která zapíná funkci trackování. Následující tabulka 4.3 shrnuje výsledky časové náročnosti. Je z ní vidět, že nejenom náhled, ale i jednoduché trackování zpomaluje běh.

`displayWindows`	`findID`	FPS s RPi kamerou	FPS s USB kamerou
`false`	`false`	7.28	4.91
`true`	`false`	6.83	3.48
`false`	`true`	5.33	5.06
`true`	`true`	5.22	3.26

Tabulka 4.3: Rychlost zpracování detekce objektů založené na prahování

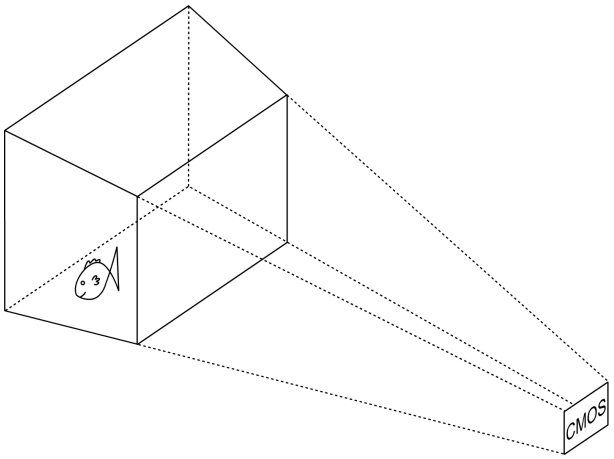
4.5 Segmentace III - Ryby

Tuto aplikaci jsem začala vyvíjet po shlédnutí reálných videí natočených pro firmu FCC Průmyslové systémy, která řešila podobný problém detekce a sledování pohybu ryb. Při schůzce s ředitelem společnosti Ing. Ottou Havlem, CSc., MBA jsem se dozvěděla něco o normálním chování ryb v nádrži a při otravě neznámým jedem. Popsal podobu akvária, ve kterém ryby žijí a jaký je účel detekce. Poté jsem konzultovala možnosti algoritmů rozpoznávání s Ing. Vojtěchem Francem, Ph.D., který mi vysvětlil problematiku modelování pozadí a popředí. Využila jsem teoretických poznatků z knihy [20] a zaměřila se na detekci pohybujících se ryb uprostřed akvária.

4.5.1 Popis problému segmentace ryb

Distributoři pitné vody používají k detekci jedů a chemikálií několik rybek v nádrži. Živé organismy mají tu výhodu, že spojitě analyzují kvalitu vody, oproti technickým senzorům, jejichž schopnost detekovat určité látky je velmi úzkopásmová [25]. Ryby dokáží detekovat celou řadu toxických látek a zajišťují tak kvalitu vody ve vodárnách. V těch je obvykle nádrž s několika rybkami (maximálně jednotky kusů), která má tvar uzpůsoben tak, aby prodloužení hran akvária mířila do ohniska kamery. Hrany pak nezpůsobují rušivé efekty, které souvisí s 3D povahou akvária (plochy v obraze netvoří čtyřúhelníky, ale jen úsečky). Mají tedy zešikmené stěny jako na Obrázku 4.10. Za akváriem je obvykle světelný zdroj, takže rybí těla tvoří v kameře tmavší regiony než je pozadí.

Běžně je akvárium sledováno operátorem, který musí kontrolovat pohyb ryb 24 hodin denně. Operátor musí sám rozhodnout, jestli jsou rybky otrávené, což závisí na jeho subjektivním vjemu a předchozích zkušenostech. Proto je snaha vyvinout automatický systém pro rozpoznávání pohybu rybiček a vyhlášení poplachu při znacích otravy, který upozorní obsluhu o možném nebezpečí.



Obrázek 4.10: Akvárium s kamerou

Znaky otravy závisí na druhu sledovaných rybiček a na typu jedu. Společné znaky se dají popsat následujícím výčtem pozorování [25]:

- plavání nahoru a dolů, ze strany na stranu
- snižování schopnosti udržet rovnováhu, převracení ryby na bok u hladiny
- klesání ke dnu otočené bokem
- ležení na dně na boku bez pohybu

Průběh otravy závisí na koncentraci jedu a na době expozice. Rybičky málokdy opustí dno akvária při běžných podmínkách v neotrávené vodě a rychlost jejich pohybu je také v určitých mezích. Při otravě začnou ryby často plavat k hladině s vysokou rychlostí a pak pozvolna klesat ke dnu. Konají rozdílný pohyb než při krmení nebo při ohrožení. V [25] jsou popsány četnosti výskytu rybek v různé hloubce v závislosti na výši koncentrace iontů kyanidu. Obecně rybky umírají po 30-60 minutách od začátku expozice. Podle [25] stačí 10 minut pro úspěšnou detekci přítomnosti jedu, takže systém stihne zareagovat dříve, než otrávená voda doputuje ke koncovým zákazníkům.

4.5.2 Rozbor řešení úlohy Ryby

Algoritmus detekce se značně liší od detekce prahováním. Použijeme metodu *background subtraction* popsané v sekci 4.2.4. Kamera se vůči akváriu nepohybuje, proto je tato metoda velmi vhodná a efektivní. Z metod zmíněných v sekci 4.2.4 a [20–22] jsem použila modelování pomocí klouzavého průměru jasových hodnot obrazových bodů. Nabízela se i metoda založená na reprezentaci Gaussovským rozdělením, která je implementována v OpenCV, nebyla vybrána z důvodu pomalejšího běhu pro real-time zpracování.

Protože se v obraze bude vyskytovat vlnění hladiny a bublinky od provzdušňovače vody, použijeme výřez obrazu, který je pro nás podstatný. Pro detekci budeme používat pouze

stupně šedi, protože barvy nehrají v detekci žádnou roli.

Jako pre-processing by bylo vhodné použít filtraci Gaussovským filtrem. Pokud ale cílíme na real-time nasazení aplikace, zbytečně prodlužuje čas zpracování jednoho snímku, proto jsem žádnou filtraci nepoužila.

Na začátku snímání nastavíme počáteční hodnoty pozadí podle rovnice 4.13. Poté si ponecháváme v paměti vždy jeden snímek pozadí (*background*) B_{t-1}.

Popředí (*foreground*) F_t získáme prahováním rozdílu snímku I_t od obrazu pozadí B_{t-1}. Tento vztah popisuje rovnice 4.22. Experimentálně jsem zvolila práh $k = 3$, aby byly vidět i nejmenší pohyby rybek za předpokladu, že nebude kolísat osvětlení akvária, a že nebude kamera dynamicky měnit clonu a čas závěrky. Tyto jevy by měly za následek, že se hodnota prahu musí pro správnou detekci zvýšit.

$$F_t = \begin{cases} 255 & \text{pro } B_{t-1} - I_t > k \\ 0 & \text{pro } B_{t-1} - I_t \leq k \end{cases} \tag{4.22}$$

Protože se ryby mohou vyskytnout i při inicializaci pozadí, nesmíme dopustit, aby vznikali duchové. To jsem zajistila právě rovnicí 4.22, kde uvažuji pouze změnu světlých pixelů na tmavší, protože ryby tvoří tmavší regiony na prosvětleném pozadí.

Aktualizaci pozadí při každém novém snímku I_t provedeme stejně jako v rovnici 4.14. Aktualizujeme pouze ty obrazové body pozadí, které nejsou překryty objekty v popředí, tedy používáme masku popředí (*foreground mask*) z diagramu 4.3. Koeficient zapomínání α jsem nastavila na 0.7, protože opět není zaručeno, že při inicializaci pozadí B_0 nebude v obraze žádný objekt, proto je potřeba zapomínat rychleji, než je obvyklé.

Na popředí dále použijeme operace eroze a dilatace se strukturním elementem z Obrázku 4.1a. Po tomto post-processingu sice zbývají malé bublinky v obraze, ale se strukturním elementem z Obrázku 4.1b nejsou ryby již spojité.

Spojité komponenty popředí nalezneme obdobně jako v aplikaci Prahování pomocí kontur. Pro získané bloby větší než prahová hodnota h spočítáme těžiště, nalezneme ohraničující rámeček a proložíme objekt elipsou. Pro výpočet použijeme teoretické znalosti ze sekce 4.2. Velikost validních blobů pro obrázek velikosti 640×480 jsem experimentálně určila na $h = 200$, menší bloby zanedbávám a odstraňuji z popředí.

Elipsa jako aproximace tvaru ryby slouží k ilustraci toho, jak rychle ryba mění svůj tvar, jak rychle se otáčí. Otrávená ryba mění svoji stopu v obraze mnohem častěji než ryba v normálním prostředí. Při otravě má ryba také náhlé křeče a prudce se mrská ze strany na stranu. Takové změny jsou zaznamenány jako prudké změny velikosti os elipsy, které lze vhodným algoritmem učení naučit, a je možné je detekovat. Takový úkol je již nad rámec této diplomové práce, proto jsem se mu nevěnovala. Pro naše další experimenty nám postačí velikost os elipsy a jejich natočení ukládat spolu s pozicí do souboru pro další zpracování.

K trackování používám stejný algoritmus jako v úloze Prahování, proto ho zde nebudu znovu připomínat.

4.5.3 Implementace úlohy Ryby

V této sekci se budu věnovat především implementaci programu, který zpracovává video soubor z disku (SD karty) a ukládá zpět do video formátu AVI s označenými objekty. Program pro RPi kameru a USB kameru s řadičem Cypress je též naimplementován, ale načítání dat z kamer již bylo řešeno v předchozích dvou úlohách.

Ryby reprezentujeme třídou `Fish`, jejíž jedna položka je třídy `Ellipse`. Obě třídy jsou definovány následovně.

```
struct Fish {
    int id;
    int area;
    float x;
    float y;
    Ellipse fitEllipse;
    Rect box;
};
struct Ellipse {
    float a;
    float b;
    float orientation;
};
```

Dále je potřeba zadefinovat konstanty použité při výpočtech. Jsou to parametr α, oblast detekce a strukturní element pro morfologické operace.

```
const float alpha = 0.7;
const Rect areaRect = Rect(10, 140, 590, 300);
const Mat mask = getStructuringElement(MORPH_CROSS, Size(3,3), Point(-1,-1));
```

Začněme otevřením videa ke čtení a inicializací zápisu videa zpět na disk. Kód je již zmíněn v sekci 4.1.2. Inicializujeme proměnné, které budeme potřebovat.

```
Mat frame;
Mat foreground;
Mat background;
vector<vector<Point> > contours;
vector<Vec4i> hierarchy;
```

První načtený snímek uložíme do proměnné pozadí. Následně ořežeme na oblast, ve které budeme hledat ryby. Poté obrázek převedeme do stupňů šedi. To samé provedeme s následujícími snímky, které ukládáme do proměnné `frameCropped`.

```
background = frame(areaRect);
cvtColor(background, background, CV_RGB2GRAY, 1);
...
Mat frameCropped;
frameCropped = frame(areaRect);
cvtColor(frameCropped, frameCropped, CV_RGB2GRAY, 1);
```

Přejdeme k realizaci rovnice 4.22. Poté provedeme post-processing ve formě dvou po sobě jdoucích erozí a dilatace.

```
subtract(background, frameCropped, foreground, noArray(), -1);
threshold(foreground, foreground, 3, 255, THRESH_BINARY);
erode(foreground, foreground, mask, Point(-1,-1), 2);
dilate(foreground, foreground, mask);
```

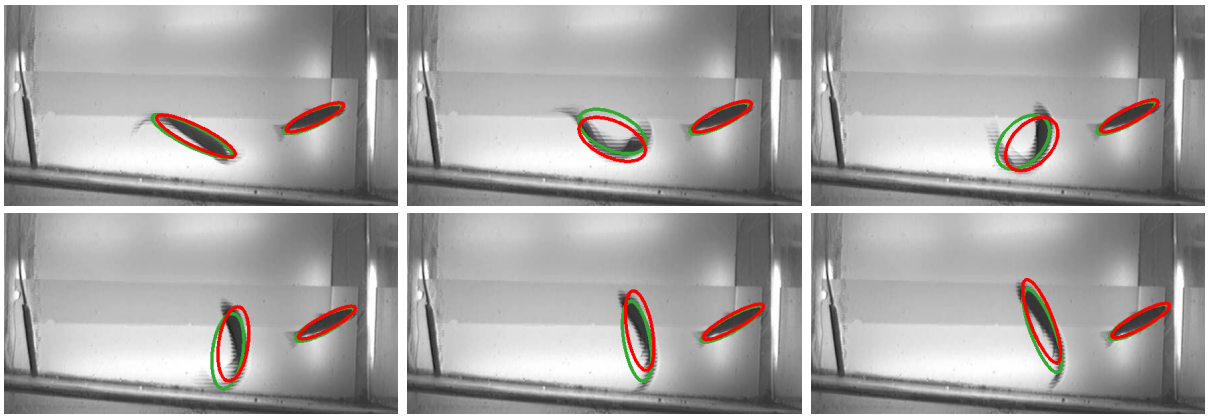
Na nalezené popředí aplikujeme stejnou funkci jako v úloze Prahování na reprezentaci kontur a nalezení spojitých komponent. Následně pro validní objekty s plochou větší než 200 spočítáme elipsu. Pro tento účel je naprogramována funkce `computeEllipse`, jejíž vstupní parametry jsou momenty spojité komponenty.

```
Ellipse computeEllipse(float mu11, float mu20, float mu02, int n)
{
    Mat covMatrix(Size(2,2),CV_32FC1);
    covMatrix = (Mat_<float>(2,2) << mu20, mu11, mu11, mu02);
    covMatrix = covMatrix/n;
    Ellipse ellipse;
    Mat eigenvalues;
    Mat eigenvectors;
    if (eigen(covMatrix, eigenvalues, eigenvectors))
    {
        ellipse.a = 2*sqrt(eigenvalues.at<float>(0,0));
        ellipse.b = 2*sqrt(eigenvalues.at<float>(0,1));
        ellipse.orientation = atan2(2*covMatrix.at<float>(0,1), ...
         ... covMatrix.at<float>(0,0) - covMatrix.at<float>(1,1))/2;
    }
    return ellipse;
}
```

Nabízí se také možnost výpočtu pomocí vestavěné funkce `fitEllipse`. Ta k výpočtu používá metodu nejmenších čtverců z [26], je ale pomalejší než moje vlastní funkce `computeEllipse`. Obě metody nevrací úplně stejné výsledky, které ilustruje následující Obrázek 4.11. Červeně je nakreslena elipsa vypočítaná z kovarianční matice, zeleně elipsa metodou nejmenších čtverců. Při malých změnách jsou elipsy velmi totožné (zelená je zakrytá červenou), při prudších změnách je rozdíl již vidět.

Nalezené objekty ukládáme do vektoru `current` a objekty z předchozího snímku jsou ve vektoru `last`. Implementace přiřazování ID je totožná s implementací z úlohy Prahování.

Pro aktualizaci pozadí v daném snímku slouží funkce `updateBackground`. Výpočet je prováděn po krocích z důvodu použití vestavěných lineárních operací s maticemi. Nejprve spočteme klouzavý průměr a uložíme do proměnné `weight`. Poté pomocí masky popředí vygenerujeme pozadí v místech, kde se nacházejí nějaké objekty. Masku invertujeme a použijeme na `weight`. Oba obrázky sloučíme a dostáváme nový model pozadí. Pomocné snímky jsou vykresleny na Obrázku 4.12.



Obrázek 4.11: Sekvence snímků ryb proložené elipsami

(a) Klouzavý průměr vně ryb (b) Původní pozadí uvnitř ryb (c) Nové pozadí

Obrázek 4.12: Aktualizace pozadí ze dvou pomocných snímků

```
Mat updateBackground(Mat& background, Mat& foreground, Mat& frame)
{
    Mat weight;
    addWeighted(background, 1-alpha, frame, alpha, 0, weight, -1);
    bitwise_and(background, foreground, background, noArray());
    bitwise_not(foreground, foreground, noArray());
    bitwise_and(weight, foreground, weight, noArray());
    add(background, weight, background, noArray(), -1);
    return background;
}
```

4.5.4 Výsledky experimentů

Univerzálnost použití algoritmu

Testovala jsem funkčnost i na videích jedoucích aut. V programu jsem upravila hodnotu koeficientu zapomínání na $\alpha = 0.1$ a změnila tvorbu masky popředí na původní vzorec z rovnice 4.15. Video totiž začíná pohledem na prázdné pozadí bez jakýkoliv rušivých objektů.

Z Obrázků 4.13 je vidět, že algoritmus je univerzální. Předpokladem použití je odladění parametrů pro danou scénu. Konkrétně pro auta ve venkovním prostředí jsem použila práh $k = 15$.

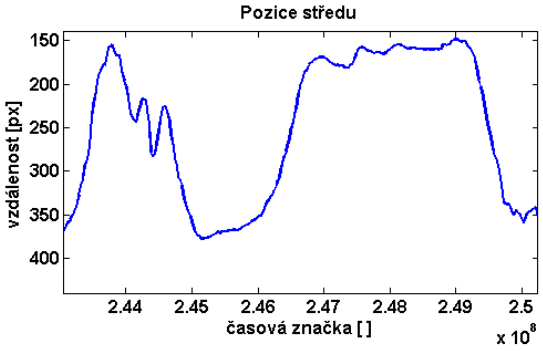
Obrázek 4.13: Sekvence snímků jedoucího auta označeného obdélníkem

Analýza měření pohybu ryb

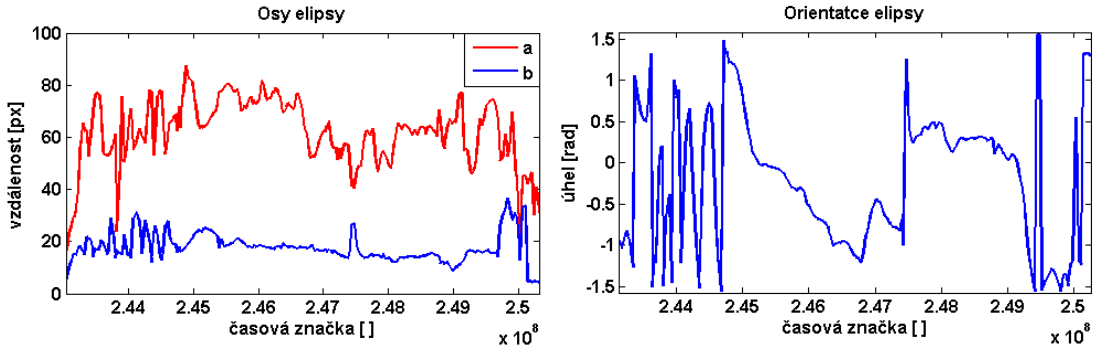
Na ukázku, jak může probíhat analýza vypočtených charakteristik objektů, jsem zanesla do grafu několik závislostí. První graf z Obrázku 4.14 zobrazuje v jaké hloubce se ryba nachází (140 = hladina, 440 = dno). Další graf z Obrázku 4.15 vykresluje velikost os proložené elipsy. Třetí graf ilustruje orientaci elipsy v rozmezí $\langle -\frac{\pi}{2}, \frac{\pi}{2} \rangle$. Na ose x je vždy časová stopa, pro představu grafy ukazují 15 s dlouhý interval pohybu již otrávené ryby.

Z grafů je poznat, že ryba plavala nejdříve ke hladině křečovitým pohybem (mění se velikost os a, b a orientace), a poté začala zvolna klesat bokem napřed (orientace se mění skoro lineárně až k úhlu 0). Pak ryba začala zase plout ke hladině, kde setrvala několik vteřin a ke konci zobrazovaného intervalu zase začala klesat ke dnu.

Takové chování je přesně popsáno v [25] a shoduje se s pozorovanými jevy v této práci. Zajímavý je ještě histogram hodnot vertikální osy, tj. hloubky, ve které se ryba nachází. V článku [25] bylo zmíněno, že při otravě se aktivita ryby přesouvá k hladině a poměr mezi hladinou a dnem je blízko 1.

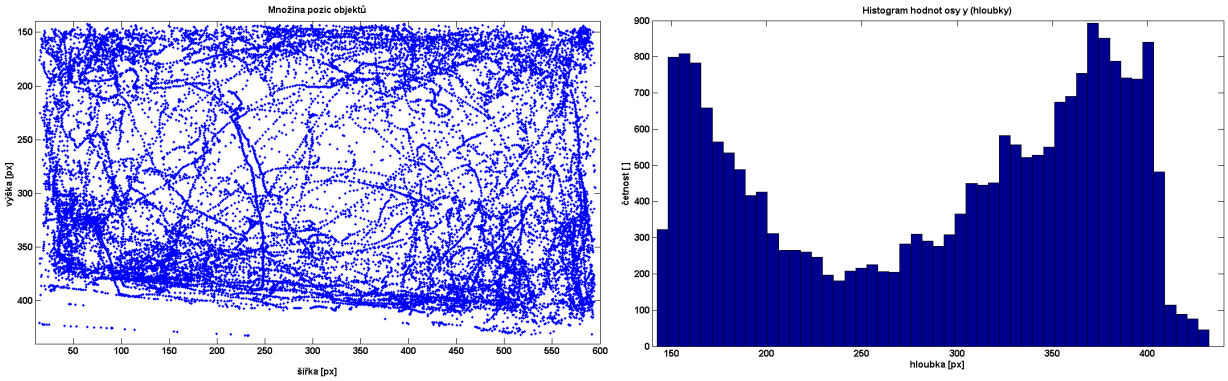


Obrázek 4.14: Závislost nalezené pozice jednoho objektu na čase



Obrázek 4.15: Závislost parametrů elipsy jednoho objektu na čase

Podívejme se na Obrázek 4.16 znázorňující všechny nalezené pozice ryb a histogram jejich hloubky. To dokazuje, že při otravě ryby často plavou k hladině.



Obrázek 4.16: Množina pozorovaných objektů a jejich histogramy

4.5.5 Návod na spuštění úlohy Ryby

V DVD příloze je uložena složka `Fish`, v níž jsou soubory

- `captureFish.cpp`
- `Makefile`
- `otrava.avi`
- `otrava_dlouha.avi`

Program se přeloží a spustí následujícím způsobem.

```
> make
> ./captureFish <video>
```

Pro spuštění programu se snímáním obrazu z kamery jsou určeny složky `Fish RPi` a `Fish USB`. Spouští se bez argumentu, tedy `./captureFish`, resp. `sudo ./BulkLoopApp`.

Na ukázku univerzálnosti řešení problému pomocí *background subtraction* je určena složka `Cars`, ve které jsou následující soubory.

- `captureCars.cpp`
- `Makefile`
- `cars1.mov`
- `cars2.mov`

Použití programu je obdobné jako u složky `Fish`.

Věnujme se teď rychlosti zpracování jednotlivých programů. V tabulce 4.4 jsou sepsány průměrné hodnoty FPS programů `Fish` a `Cars`, které zpracovávají video. Programy byly spuštěny na notebooku Dell s procesorem Intel Core2Duo 2.2 GHz, 4 GB RAM ve virtualizovaném operačním systému Debian 7.4 Wheezy programem VirtualBox.

program	Fish	Cars
rozlišení videa	640×480	848×480
velikost oblasti detekce	460×300	698×380
naměřené FPS zpracování	30.568	21.554

Tabulka 4.4: Srovnání výkonnosti zpracování snímků založené na modelu pozadí programů `Fish` a `Cars`

V Tabulce 4.5 je shrnut průměrný počet snímků zpracovaných za sekundu programem určeným pro RPi kameru s různě nastavenými parametry. Rozlišení snímků je 640×480 pixelů se srovnatelným výřezem zajímavé oblasti 460×300 pixelů kvůli porovnání s programem `Fish`. Záměrně zde neuvádím hodnoty pro USB kameru, protože ji vložený kód detekce natolik zdržoval, že se program často zasekával nebo se začaly ztrácet pakety.

`displayWindows`	`findID`	FPS s RPi kamerou
`false`	`false`	7.36
`true`	`false`	3.58
`false`	`true`	5.57
`true`	`true`	3.37

Tabulka 4.5: Srovnání výkonnosti zpracování snímků založené na modelu pozadí pro RPi kameru

Kapitola 5

Závěr

Seznámila jsem se s modulem Raspberry Pi, s operačním systémem Raspbian a knihovnou OpenCV. Sestavila jsem z Raspberry Pi a Raspberry Pi Camera Board měřicí kameru, která je umístěná v praktické plastové krabičce, je lehká a snadno přenositelná. K modulu lze připojit i jiné USB kamery nebo webkamery. V této práci jsem dále použila kameru s řadičem Cypress a monochromatickým CMOS senzorem od Micronu.

Ověřila jsem, že se dá modul Raspberry Pi použít pro videometrické účely na Katedře měření FEL, ČVUT pro své vysoké rozlišení a relativně malé zkreslení. S přihlédnutím k faktu, že Raspberry Pi je ve skutečnosti minipočítač s relativně vysokým výkonem vzhledem k velikosti a ceně, a umožňuje mnoho dalších aplikací jako stolní počítače, bude jeho využití doslova od A do Z.

V této práci jsem se zabývala návrhem a vývojem demonstračních aplikací, které lze dále rozvíjet a vylepšovat a přispět tak k vědecké činnosti laboratoře videometrie. V práci používám mnohé poznatky z oblasti počítačového vidění a zpracování obrazu.

První dvě aplikace s pracovními názvy Laser a Prahování byly vyvinuty během prvního seznamování s modulem Raspberry Pi, programování v jazyce C++ pod Linuxem a knihovnou OpenCV. Proto se zabývají základními metodami zpracování obrazu.

Třetí aplikace vnáší do této práce kousek reálného problému kontroly čistoty a nezávadnosti vody pomocí detekce pohybu ryb. Měla jsem na výběr implementaci dvěma algoritmy modelování pozadí, vybrala jsem si ale ten se snazší implementací a rychlejším během na Raspberry Pi. Pro použití *background subtraction* na detekci rybiček v akváriu jsem musela použít neobvyklé hodnoty parametrů pro výpočet modelu pozadí, protože se ryby vždy nacházejí v obraze a nebylo možné model naučit na počátečních snímcích.

Jinak je tomu u aplikace pro zpracování videí, zachycující scénu se statickým obrazem bez objektů v popředí v prvních snímcích videa. Tento program jsem nazvala Cars a vyzkoušela s obvyklým výpočtem modelu pozadí. Tímto algoritmem pak můžeme segmentovat pozadí od popředí v jakékoliv konfiguraci prostředí se statickým pohledem na scénu.

Reprezentace rybiček jako aproximace elipsou se osvědčila a získanými hodnotami parametrů ryb při otravě jsme potvrdili výsledky článku [25]. V této práci jsem se ale dále

nezabývala hledáním znaků otravy v získaných datech.

Protože cílem práce bylo použít měřicí kameru pro zpracování obrazu, otestovala jsem rychlost zpracování implementovaných programů. Získané hodnoty zpracování průměrného počtu snímků za sekundu nebyly podle mého očekávání. Přesto musím za svými výsledky stát, protože se mi podařilo během implementace rychlost zpracování vždy o něco zvýšit optimalizací použitých proměnných nebo zbytečných výpočtů. S ohledem na použitý hardware a nízkou pořizovací cenu zařízení musím vývojáře z Raspberry Pi Foundation pochválit.

Je důležité říci, že kontrola kvality vody je nezbytný bezpečnostní prvek ve vodárně a jistě by nikdo ani nepomyslel, že zařízení za 35$ bude dohlížet na kvalitu pitné vody pro tisíce až statisíce obyvatel, proto považuji výsledek mé práce za velmi uspokojivý a pro demonstrační účely jistě postačí.

Na závěr sem patří shrnutí poznatků o Raspberry Pi, které jsem posbírala během práce na tomto zadání. Raspberry Pi jsem si velmi oblíbila, protože přesně takový kus hardware tu chyběl. Jistě každého napadne zařízení Arduino, ale Raspberry Pi má navíc operační systém, který se chová v podstatě úplně stejně, jako kterýkoliv osobní počítač s Debianem, je jen o něco pomalejší. Proto bych Raspberry Pi doporučila všem, kteří si chtějí hrát s automatizací a senzory, ale nevědí jak na to. Je však třeba počítat s omezeným výkonem, který je daň za operační systém. To považuji za jedinou slabinu Raspberry Pi.

Seznam obrázků

Seznam tabulek

Literatura

[1] Severance, C.: *Eben Upton: Raspberry Pi*, IEEE Computer Magazine, Volume 46, October 2013. str. 14-16.

[2] Raspberry Pi Foundation: *Raspberry Pi Foundation home page.* [cit. 27.12.2013]. `http://www.raspberrypi.org`

[3] Soukup T.: *Raspberry Pi: Už dva miliony prodaných kusů, dříve než se čekalo*, Živě.cz, 18.11.2013. [cit. 27.12.2013]. `http://www.zive.cz/bleskovky/raspberry-pi-uz-dva-miliony-prodanych-kusu-drive-nez-se-cekalo/sc-4-a-171369/default.aspx`

[4] Akerman, D.: *Raspberry Eye In The Sky*, blog, 27.5.2013. [cit. 13.4.2014]. `http://www.daveakerman.com/?p=1154`

[5] Raspberry Pi Foundation: *Raspberry Pi Foundation Hardware Documentation.* [cit. 2.5.2014]. `http://www.raspberrypi.org/documentation/hardware/README.md`

[6] Raspberry Pi Wiki: *Raspberry Pi Wiki Hub.* [cit. 27.12.2013]. `http://www.elinux.org/RPi_Hub`

[7] GitHub: *Jay Rambhia main page.* [cit. 27.12.2013]. `https://github.com/jayrambhia`

[8] OmniVision OV5647: *Product brief.* [cit. 27.12.2013]. `http://www.ovt.com/download_document.php?type=sensor&sensorid=66`

[9] OmniVision OV5647: *Datasheet.* [cit. 27.12.2013]. `http://www.seeedstudio.com/wiki/images/3/3c/Ov5647_full.pdf`

[10] CSI specification: *Camera interface.* [cit. 27.12.2013]. `http://www.mipi.org/specifications/camera-interface`

[11] Linux projects: *UV4L driver for RPi.* [cit. 27.12.2013]. `http://www.linux-projects.org/modules/sections/index.php?op=viewarticle&artid=14`

[12] Raspberry Pi Forum: *Official V4L2 driver* [cit. 3.1.2014].
`http://www.raspberrypi.org/forums/viewtopic.php?f=43&t=62364&sid=c9c6d2ff060d3e05d530d93eed48cbb0`

[13] Zoubek, M.: *Určování polohy pomocí řádkových obrazů*, Diplomová práce, FEL, ČVUT, 2011.

[14] Micron MT9M001: *1/2-Inch Megapixel Digital Image Sensor Features*, Datasheet, Micron, 2004.

[15] Cypress CY7C68013A: *EZ-USB FX2LP USB Microcontroller High-Speed USB Peripheral Controller*, Datasheet, Cypress Semiconductor Corporation, 2013.

[16] Fischer, J.: *Optoelektronické senzory a videometrie*, Skriptum, ČVUT, FEL, 2002. ISBN 80-01-02525-X

[17] Ondrička, M.: *Rýchly vstup dát po zbernici USB do Raspberry Pi*, Bakalářská práce, FEL, ČVUT, 2014.

[18] OpenCV: *OpenCV 2.4.9.0 Documentation*. [cit. 2.5.2014].
`http://http://docs.opencv.org/`

[19] Hlaváč, V.; Sedláček, M.: *Zpracování signálů a obrazů*, Skriptum, ČVUT, FEL, 2009. ISBN 978-80-01-04442-1

[20] Šonka, M.; Hlaváč V.; Boyle R.: *Image Processing, Analysis and Machine Vision*, Thomson, third edition, 2008. ISBN 0-495-08252-X

[21] Parks, D. H.; Fels, S. S.: *Evaluation of Background Subtraction Algorithms with Post-Processing*, IEEE 5^{th} International Conference on Advanced Video and Signal Based Surveillance, 2008. str 192-199.

[22] Mohamed, S. S.; Tahir, N. M.; Adnan, R.: *Background modelling and background subtraction performance for object detection*, The 6^{th} International Colloquium on Signal Processing and Its Applications (CSPA), 2010. str. 1-6.

[23] KadewTraKuPong P.; Bowden R.: *An improved adaptive background mixture model for real-time tracking with shadow detection*, Proceedings of 2^{nd} European Workshop on Advanced Video-Based Surveillance Systems, 2001. str. 1-5.

[24] Kléma, J.: *Shluková analýza - formalizace, základní algoritmy*, přednášky předmětu Strojové učení a analýza dat, ČVUT, FEL, 2013.

[25] Nogita, S.; Baba, K.; Yahagi, H.; Watanabe, S.; Mori, S.: *Acute toxicant warning system based on a fish movement analysis by use of AI concept*, Proceedings of the International Workshop on Artificial Intelligence for Industrial Applications, 1988. str. 273-276 .

[26] Fitzgibbon, A. W.; Fisher, R. B.: *A Buyer's Guide to Conic Fitting*, Proceedings of 5th British Machine Vision Conference, Birmingham, 1995. str 513-522.

Příloha A

Obsah DVD

- Aplikace - Složky se zdrojovými kódy a ukázkovými videi
  - Cars
  - Fish
  - Fish RPi
  - Fish USB
  - Laser RPi
  - Laser USB
  - Thresh RPi
  - Thresh USB
  - Test OpenCV
  - Test RPi
  - Test USB
- Dokumentace - Obsahuje tuto práci ve formátu PDF
- Literatura - Odborná literatura, ze které jsem čerpala. Značení podle kapitoly 5.