Fakulta elektrotechnická
Katedra elektroniky a informačních technologií

Bakalářská práce

Použití FLIR Lepton modulu

Autor práce: Jan Pillmann
Vedoucí práce: Ing. Petr Weissar, Ph.D.

Plzeň 2022

ZÁPADOČESKÁ UNIVERZITA V PLZNI
Fakulta elektrotechnická
Akademický rok: 2021/2022

ZADÁNÍ BAKALÁŘSKÉ PRÁCE
(projektu, uměleckého díla, uměleckého výkonu)

Jméno a příjmení:	**Jan PILLMANN**
Osobní číslo:	**E19B0106P**
Studijní program:	**B2612 Elektrotechnika a informatika**
Studijní obor:	**Elektronika a telekomunikace**
Téma práce:	**Použití FLIR Lepton modulu**
Zadávající katedra:	**Katedra elektroniky a informačních technologií**

Zásady pro vypracování

Vytvořte vzorové aplikační řešení FLIR modulu Lepton

1. Aplikace běžící na PC a komunikující prostřednictvím USB modulu
2. Aplikace běžící na mikrokontroléru (ARM) s přímou komunikací s HW modulu Lepton

Rozsah bakalářské práce: **30 – 40**
Rozsah grafických prací: **dle doporučení vedoucího**
Forma zpracování bakalářské práce: **elektronická**

Seznam doporučené literatury:

The FLIR Lepton [online]. [cit. 2021-04-13]. Dostupné z: https://lepton.flir.com/ *LWIR Micro Thermal Camera Module Lepton* [online]. [cit. 2021-04-13]. Dostupné z: https://www.flir.eu/products/lepton/

Vedoucí bakalářské práce: **Ing. Petr Weissar, Ph.D.**
Katedra elektroniky a informačních technologií

Datum zadání bakalářské práce: **8. října 2021**
Termín odevzdání bakalářské práce: **26. května 2022**

Západočeská univerzita v Plzni
Fakulta elektrotechnická

L.S.

Prof. Ing. Zdeněk Peroutka, Ph.D.
děkan

Doc. Ing. Jiří Hammerbauer, Ph.D.
vedoucí katedry

V Plzni dne 8. října 2021

Abstrakt

Práce se zabývá problematikou použití moderních modulů termální kamery Lepton od firmy FLIR. Tyto moduly mají širokou oblast využití díky jejich univerzálnosti, malé velikosti, nízké spotřebě a cenové dostupnosti. Cílem bylo vytvořit a popsat komunikační rozhraní mezi modulem a počítačem, k čemuž byl použit mikrokontrolér STM32 naprogramovaný v jazyce C a počítačová aplikace pro zpracování a grafické zobrazení dat v jazyce C#. Výsledkem řešení je popis komunikace s kamerovým modulem a ukázkové příklady algoritmů, na kterých jsou principy vysvětleny. Dále jsou stanoveny požadavky na časovou synchronizaci a přenosová rozhraní mezi jednotlivými částmi systému. Výsledky lze upravit a použít pro vlastní řešení využívající tento kamerový modul.

Klíčová slova

FLIR, Lepton, Termální kamera, Infračervené měření teploty, STM32, Detekce pohybu, VoSPI packet, Palety barev, Greyscale, Ironbow, Rainbow

Abstract

Pillmann, Jan. *Application of FLIR Lepton module* [*Použití FLIR Lepton modulu*]. Pilsen, 2022. Bachelor thesis (in Czech). University of West Bohemia. Faculty of Electrical Engineering. Department of Electronics and Information Technologies. Supervisor: Petr Weissar

---

This thesis aims to analyze the use of modern thermal camera modules Lepton by the FLIR Company. These modules can be widely used due to their universality, small size, low power consumption, and affordability. The goal was to create and describe a communication interface between the module and the computer, using an STM32 microcontroller programmed in C and a computer application for data processing and graphical display in C#. The solution consists of a description of communication with the camera module and example algorithms with an explanation of the principles. Timing and communication interface requirements are specified for each part of the system. The results can be modified and used as a solution to a particular use of this camera module.

Keywords

FLIR, Lepton, Thermal camera, Infrared temperature measurement, STM32, Movement detection, VoSPI packet, Color pallets, Greyscale, Ironbow, Rainbow

Prohlášení

Předkládám tímto k posouzení a obhajobě bakalářskou práci, zpracovanou na závěr studia na Fakultě elektrotechnické Západočeské univerzity v Plzni.

Prohlašuji, že jsem svou závěrečnou práci vypracoval samostatně pod vedením vedoucího bakalářské práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury na konci práce. Jako autor uvedené bakalářské práce dále prohlašuji, že v souvislosti s vytvořením této závěrečné práce jsem neporušil autorská práva třetích osob, zejména jsem nezasáhl nedovoleným způsobem do cizích autorských práv osobnostních a jsem si plně vědom následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., včetně možných trestněprávních důsledků vyplývajících z ustanovení § 270 trestního zákona č. 40/2009 Sb.

Také prohlašuji, že veškerý software, použitý při řešení této bakalářské práce, je legální.

V Plzni dne 24. května 2022

Jan Pillmann

.....

Podpis

Obsah

Seznam obrázků

Seznam symbolů a zkratek

SPI	Serial peripheral interface
MOSI	Master out, slave in
MISO	Master in, slave out
CS	Chip select
UART	Universal asynchronous receiver-transmitter
SDK	Software development kit
HAL knihovna	Hardware abstraction layer library
CMSIS	Common microcontroller software interface standard
PNG	Portable network graphics

1

Úvod

Bezkontaktní měření teploty těles lze použít pro automatické získávání dat o objektech ve světě, které by s jinými technologiemi bylo získat složité, ne-li nemožné. Je založeno na principu měření množství vyzářené energie infračerveného záření, které emitují teplá tělesa. Termokamery využívají tento princip a teplotu měří pro matici několika bodů vedle sebe. Vzniká tím obraz, popisující rozložení teploty ve snímaném prostředí, ze kterého lze jednoduše vyčíst kontrast teplot okolních bodů. Data je možné použít pro celou škálu aplikací, jako je například sledování pohybu lidí a zvířat, prevence požáru, zjištění umístění předmětů a mnoho dalších.

S postupujícím vývojem termokamer vznikají kamerové moduly, které díky své malé velikosti, nízké spotřebě a cenové dostupnosti je možné integrovat do vestavěných zařízení, která následně automaticky sbírají a zpracovávají data. Univerzalita řešení spočívá ve velkém výběru z možností napájení a komunikačních rozhraní pro předávání dat. Nízký odběr energie umožňuje využít různé typy zdrojů napájení, jako je například síť, baterie nebo solární článek. Předávání dat ze zařízení probíhá skrze libovolné dostupné komunikační rozhraní, které musí splňovat několik podmínek, které jsou stanoveny v následujících kapitolách.

Cílem této bakalářské práce je vytvořit a popsat komunikaci mezi mikrokontrolérem z rodiny STM32 a termokamerovým modulem Lepton od firmy FLIR. Následně přenos dat z mikrokontroléru do aplikace běžící v počítači, která obraz zpracuje a graficky zobrazí. Výsledné řešení popisuje obecný princip použití těchto technologií a je možné upravit ho pro řešení vlastního problému.

2

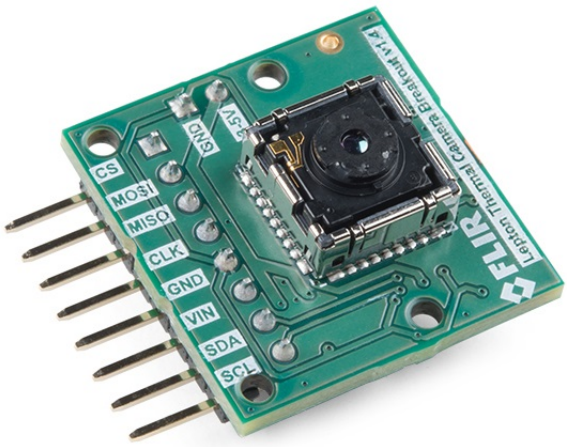
FLIR Lepton

2.1 Charakteristika modulu

Lepton označuje celou řadu modulů termálních kamer od firmy Flir. Jedná se o moduly o velikosti 11,8 x 12,7 x 7,2 mm s hmotností 0,9 g, díky tomu jsou vhodné pro integraci do zařízení s širokým spektrem využití. Pro napájení je potřeba 2,8 V, 1,2 V a 2,5 až 3,1 V pro vstupně výstupní vývody. Odebíraný příkon je 800 mW při ovládání závěrky, kdy probíhá kalibrace jednotlivých obrazových bodů. Při aktivním snímání je příkon 160 mW a ve standby režimu klesne příkon na 5 mW. Připojení k modulu je zajištěno přes Molex patici se 32 vývody.

Snímková frekvence je omezena na 8,7 Hz, aby byly dodrženy omezení na maximální snímkovou frekvenci exportovaného zboží ze země výrobce. Minimální teplotní citlivost obrazových bodů lze očekávat 50 mK (0,05 °C).

Pro účely této bakalářské práce byla použita vývojová deska Lepton Thermal Camera Breakout v1.4 (Obr. 2.1). Deska obsahuje uvedenou patici pro připojení modulu, jehož vývody pro komunikaci a napájení vyvádí na 2,54mm pin header a zároveň vytváří všechny potřebné napájecí úrovně z jednoho 3V až 5V zdroje napětí.



Obr. 2.1: Použitá vývojová deska kamerového modulu při vývoji |Převzato z [3]|

2.2 Porovnání verzí

Lepton je dostupný v několika verzích, které se liší hlavně v rozlišení a v dostupných funkcích. Mezi funkce patří hlavně radiometrie, která zajišťuje kalibrovaná měření absolutní teploty v jednotlivých obrazových bodech. Bez této funkce jsou získaná data pouze relativním poměrem mezi bodem s nejvyšší a nejnižší teplotou scény.

2.2.1 Verze 1.5, 1.6 a 2.0

Všechny tyto tři verze mají společné rozlišení 80 x 60 pixelů a nepodporují radiometrii. Horizontální zorné pole činí 50 ° s výjimkou verze 1.6, která má zorné pole 25 °. Od verze 2.0 mají všechny modely závěrku.

2.2.2 Verze 2.5

Jedná se o vylepšenou verzi 2.0, která má totožné parametry, až na to, že podporuje radiometrii. Tato verze modulu byla použita při softwarovém vývoji v této bakalářské práci a jsou na ní vysvětleny všechny postupy.

2.2.3 Verze 3.0

Třetí verze modulu zvyšuje rozlišení na 160 x 120 pixelů a také zvyšuje horizontální zorné pole na 57 °.

2.2.4 Verze 3.5

Obdobně, jako u verze 2.0 a 2.5, je tato verze identická s verzí 3.0, má ale navíc podporu radiometrie.

2.3 Telemetrie

Telemetrie je volitelný režim, který k obrazovým datům připojí dodatečné informace, jako je například verze softwaru, sériové číslo, uplynulý čas od spuštění modulu, počet zaznamenaných snímků, teplota snímacího čipu, teplota pouzdra, aktuální hodnoty kalibrace emisivity, ... [4].

2.4 Formáty dat pořízených snímků

2.4.1 Raw14

Modul nabízí dva výstupní formáty dat. Prvním z nich je Raw14, kdy je každý pixel vyjádřen 14bitovou hloubkou odstínů šedé barvy, kde nižší hodnoty představují body s nižší teplotu a vyšší hodnoty představují body s vyšší teplotou. Pokud je zapnutý režim radiometrie, hodnoty přímo vyjadřují absolutní teplotu v Kelvinech vynásobenou konstantou pro zahrnutí desetinné části čísla v binární reprezentaci. Pomocí rovnice 2.1 jsou naměřená data přepočítána na stupně Celsia:

$$t = \frac{x}{k} - 273,15 \quad [^{\circ}C] \tag{2.1}$$

kde x je naměřená hodnota a k je konstanta pro převod desetinného místa. Ve výchozí konfiguraci modulu je konstanta k nastavena na 100, pokud by naměřená hodnota x byla například 305, teplota ve stupních Celsia t je vypočítána po dosazení do rovnice 2.1 následovně v rovnici 2.2:

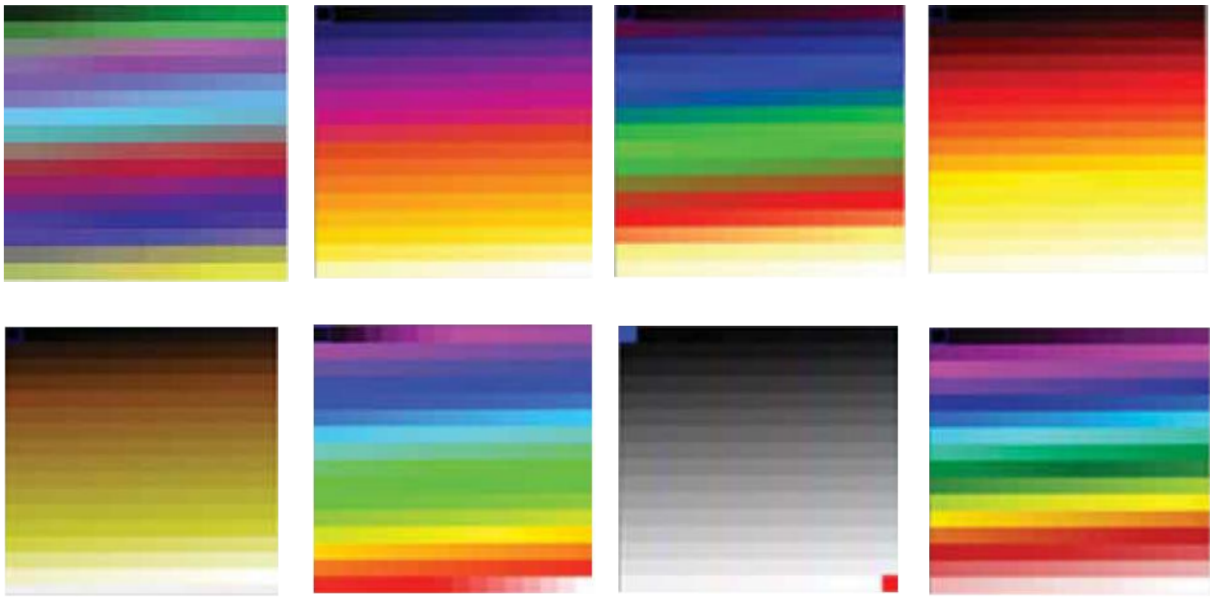
$$t = \frac{30050}{100} - 273,15 = 27,35 \quad [^{\circ}C] \tag{2.2}$$

Pokud modul nepodporuje radiometrii nebo ji nemá zapnutou, naměřené hodnoty jsou automaticky škálovány mezi nejstudenější a nejteplejší bod scény. Výsledkem je normalizovaný černobílý obraz, který lze použít i bez algoritmu pro vlastní převedení dat do grafické podoby. Avšak v tomto režimu není možné zjistit teplotu jednotlivých bodů, jelikož data jsou pouze relativní k ostatním bodům scény.

V této bakalářské práci byl použit formát Raw14 se zapnutou radiometrií. Pro konverzi snímků do podoby vhodné ke grafickému zobrazení je dále vyvinut vlastní algoritmus. Přepočet ale probíhá až nad získanými absolutními teplotami, tato informace o teplotě tedy není ztracena a je možné k ní nadále přistupovat při vyhodnocování dat.

2.4.2 RGB888

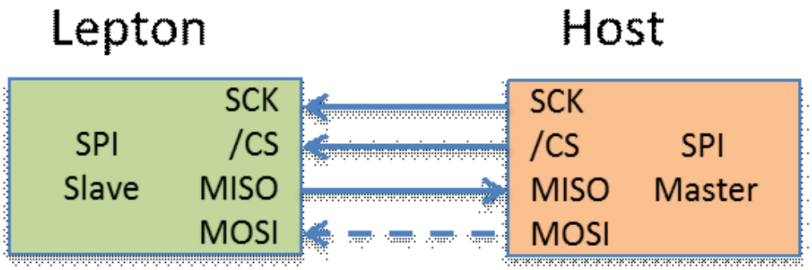
Druhým podporovaným výstupním formátem dat je RGB888, který naměřená data normalizuje a aplikuje na ně paletu barev. Tyto palety je jednak možné uživatelsky definovat a nahrát do modulu nebo si vybrat z osmi předem připravených v modulu (Obr. 2.2).



Obr. 2.2: Předdefinované palety barev v kamerovém modulu |Převzato z [4]|

2.5 Komunikace

Lepton používá dvě komunikační rozhraní, SPI pro hlavní datový přenos naměřených dat a I^2C pro dodatečnou konfiguraci parametrů modulu. Po SPI rozhraní probíhá komunikace pouze v jednom směru (z kamery do čtecího zařízení), jsou tedy potřeba pouze 3 vodiče (zapojení viz Obr. 2.3). V této práci se pracuje s modulem ve výchozí konfiguraci, a proto není rozhraní I^2C použito.



Obr. 2.3: SPI rozhraní kamerového modulu. |Převzato z [4]|

Veškerá datová komunikace obrazových dat probíhá pomocí tzv. VoSPI paketů přenášených skrze SPI rozhraní. Čtecí zařízení pracuje v režimu master a kamerový modul v režimu slave. Komunikace nevyžaduje žádné speciální časovací a synchronizační signály, celý komunikační proces řídí master zařízení. Data z Leptonu lze číst proměnnou rychlostí (frekvence CLOCK signálu), díky tomu existuje více možností, jak vytvořit čtecí algoritmus a z toho odvíjející se časování celého programu.

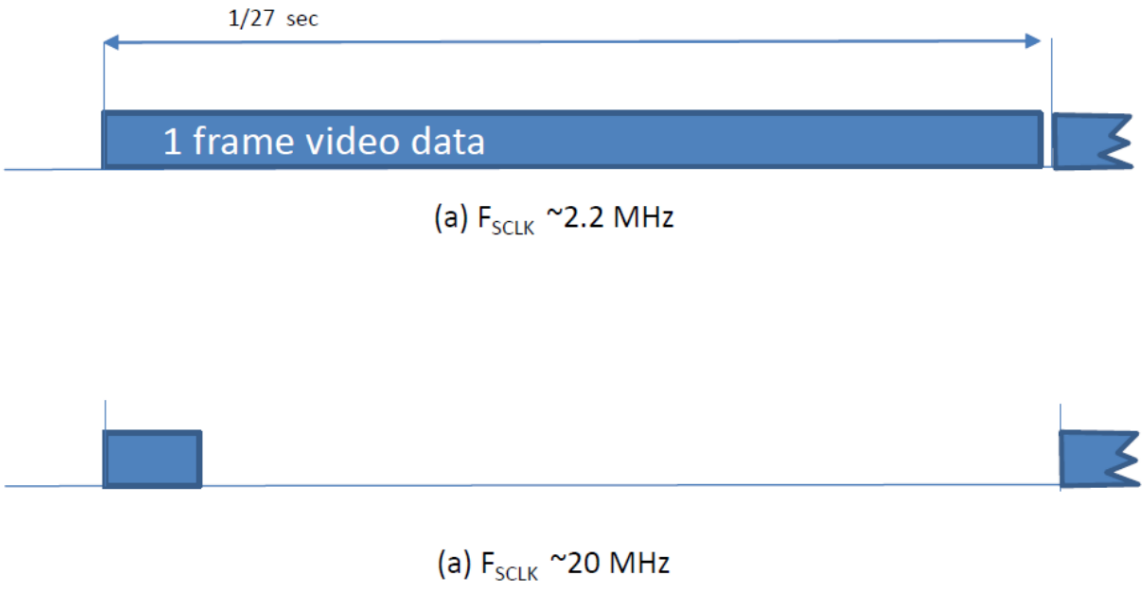
Kamerový modul odesílá 27 snímků za sekundu a všechny snímky je nutno přečíst, jinak modul vyhodnotí ztrátu synchronizace a restartuje se. Z výpočtu periody v rovnici 2.3:

$$T = \frac{1}{f} = \frac{1}{27} \doteq 37 \quad [ms] \tag{2.3}$$

je patrné, že na přečtení jednoho snímku je čas pouze okolo 37 ms. V tomto čase se musí snímek nejen přečíst, ale zároveň se musí provést veškeré zpracování dat. Díky nízkým nárokům na rychlost komunikace je možné buď snímek přečíst za co nejkratší čas, modul poté odesílá discard pakety, než je další snímek připraven. Nebo je možné naopak využít celou délku 37 ms a snímek číst s nižší rychlostí (Obr. 2.4). Minimální potřebná frekvence CLK signálu f_{CLKmin} je při rozlišení 80 x 60 obrazových bodů s 16bitovou hloubkou černobílého odstínu a 27 snímcích za sekundu vypočtena z rovnice 2.4:

$$f_{CLKmin} = 80 \cdot 60 \cdot 16 \cdot 27 \doteq 2,074 \quad [MHz] \tag{2.4}$$

Ve výpočtu je použito 16 bitů pro barevnou hloubku, přestože kamerový modul snímá pouze ve 14 bitech. Je to z toho důvodu, protože hodnota je vždy zarovnána na 2 byty a zbylé 2 nejvyšší bity mají úroveň logické 1. Nejvyšší možná frekvence je $f_{CLKmax} = 20\ MHz$ a je stanovena v datovém listu [4].



Obr. 2.4: Dva možné způsoby čtení snímků z modulu. |Převzato z [4]|

2.5.1 VoSPI paket

VoSPI paket je minimální datový objekt, kterým kamerový modul komunikuje s master zařízením. Skládá se z identifikačního čísla (ID) paketu (2 byty), kontrolního součtu CRC (2 byty) a zbytek tvoří obrazová data, jejichž velikost závisí na zvoleném formátu. Pro Raw14 jsou data 160bytová, pro RGB888 jsou 240bytová (jeden pixel je tvořen 3 byty místo 2).

ID hodnota slouží pro dva účely. Do jednoho paketu se vejde jeden horizontální řádek obrazových dat, a aby master zařízení rozpoznalo, který řádek přijímá, ID hodnota obsahuje pořadové číslo tohoto aktuálně přenášeného řádku. Následně, když je celý obraz přenesen, ID počítadlo se resetuje a začíná zase od nuly. Druhý případ je, když ID určuje, že právě dochází k synchronizaci a nebo že se všechna dostupná data již stihla přečíst a momentálně je nutno počkat, než se připraví další řádek. V tomto případě má ID v hexadecimálním tvaru hodnotu xFxx, kde x značí libovolnou hodnotu. Tyto pakety se nazývají discard pakety a obrazová data neobsahují užitečné informace.

Verze leptonu s rozlišením 80 x 60 odesílá na jeden snímek 60 paketů, verze s rozlišením 80 x 120 odesílá paketů 120. Při povolené telemetrii (podkapitola 2.3) je v obou případech počet paketů vyšší, v této práci ale telemetrie není zapnuta.

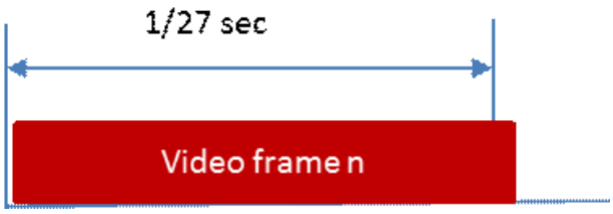
2.5.2 Opakované snímky

Kamerový modul sice odesílá přibližně 27 snímků za sekundu, jak již ale bylo zmíněno, počet unikátních snímků je omezen přibližně na 8,7 snímků za sekundu, aby byly dodrženy omezení na maximální snímkovou frekvenci exportovaného zboží ze země výrobce. Vychází to tak, že na každý třetí snímek spadají 2 zopakované. Při návrhu softwaru je toto chování možné zohlednit a duplicitní snímky dále nezpracovávat a ušetřit tak instrukční čas.

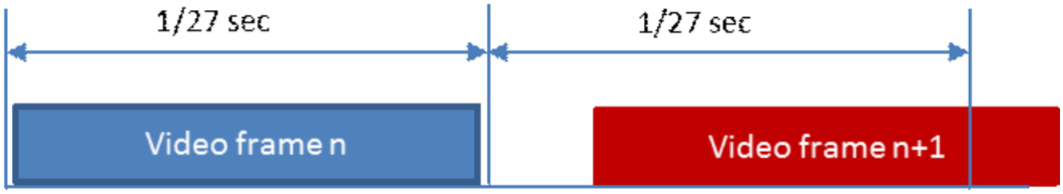
Kamera odesílá všech 27 snímků za sekundu z důvodu, aby bylo rozhraní mezi systémy standardizované pro parametry běžných typů kamer.

2.5.3 Desynchronizace

Pokud se z různých důvodů nepovede přečíst celý snímek přibližně za 37 ms (viz rovnice 2.3), Lepton vyhodnotí, že nastala desynchronizace a spustí proces nové synchronizace, kdy zahodí aktuální data a odesílá discard pakety. Jednotlivé případy jsou uvedeny na Obr. 2.5, 2.6 a 2.7.

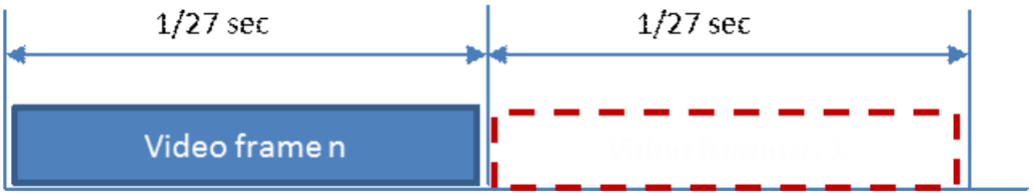


Obr. 2.5: Desynchronizace způsobena příliš pomalým čtením o nízké frekvenci clock signálu f_{CLK} |Převzato z [4]|



Obr. 2.6: Desynchronizace způsobena prodlevou před začátkem čtením následujícího snímku. |Převzato z [4]|

Následkem přidaného zpoždění (Obr. 2.6) je, že snímek není celý přečten ve stanovené době. K tomuto může dojít například prodlevou softwarového zpracování předchozího snímku.



Obr. 2.7: Desynchronizace způsobena nepřečtením následujícího snímku. |Převzato z [4]|

3

Komunikace ARM mikrokontroléru s Lepton modulem

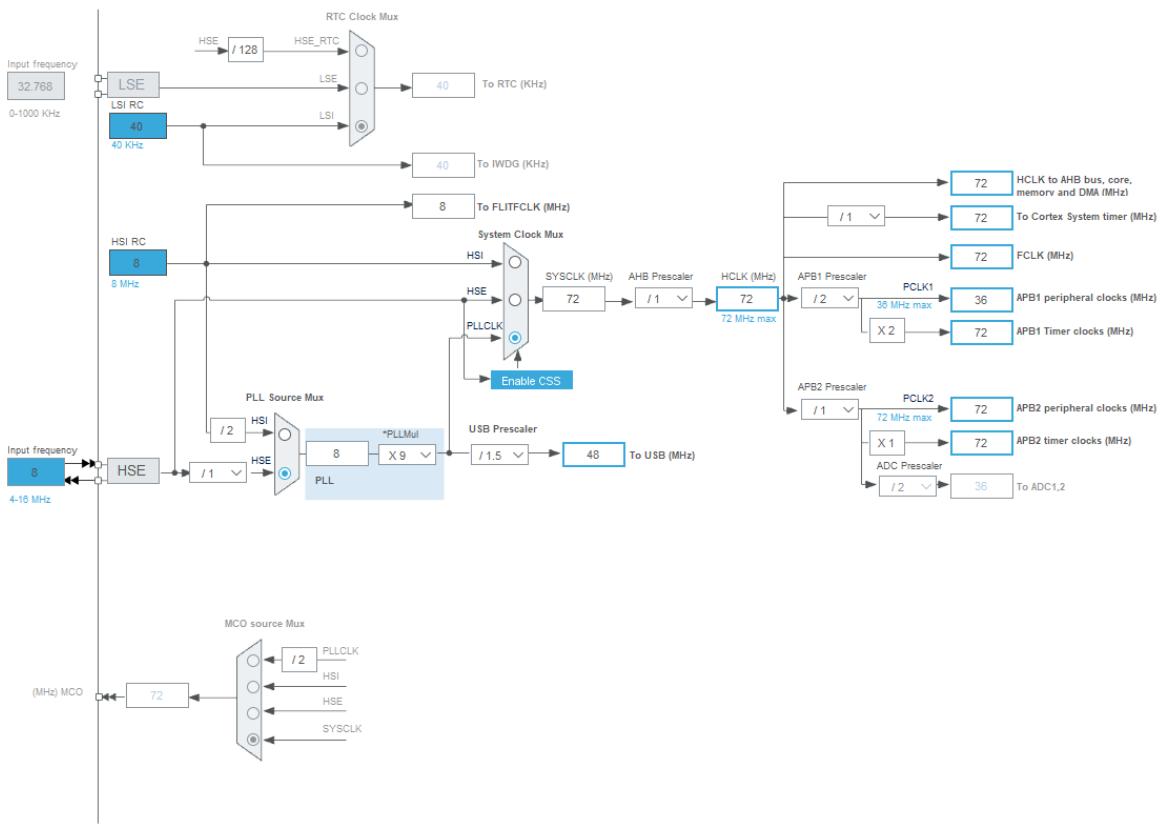
Pro komunikaci s kamerovým modulem Lepton byl použit mikrokontrolér STM32 od firmy STMicroelectronics, a to vývojový modul Blue Pill s osazeným mikrokontrolérem STM32F103C8. Dále bylo použito vývojové prostředí STM32CubeIDE pro správu projektu, konfiguraci periferií mikrokontroléru, kompilaci kódu a nahrání programu. Samotné psaní a formátování kódu probíhalo v prostředí Visual Studio Code. Pro verzování kódu byl použit nástroj Git.

3.1 Konfigurace STM32 mikrokontroléru

Konfiguraci a obsluhu periferií mikrokontroléru zajišťuje použitá HAL knihovna. Kmitočty všech periferií a zejména jádra procesoru jsou nastaveny na maximální hodnotu (Obr. 3.1), aby bylo možné zpracovat relativně vysoké množství dat získaných z kamery.

SPI rozhraní pro komunikaci s kamerou je nastaveno na frekvenci $f_{CLK} = 18$ Mb/s, polarita clock signálu CPOL = 0 a fáze CPHA = 1. Nastavení CPOL odporuje stanovené hodnotě v datovém listu kamery, během vývoje bylo zjištěno, že daná konfigurace nefunguje a hodnota musí být opačná.

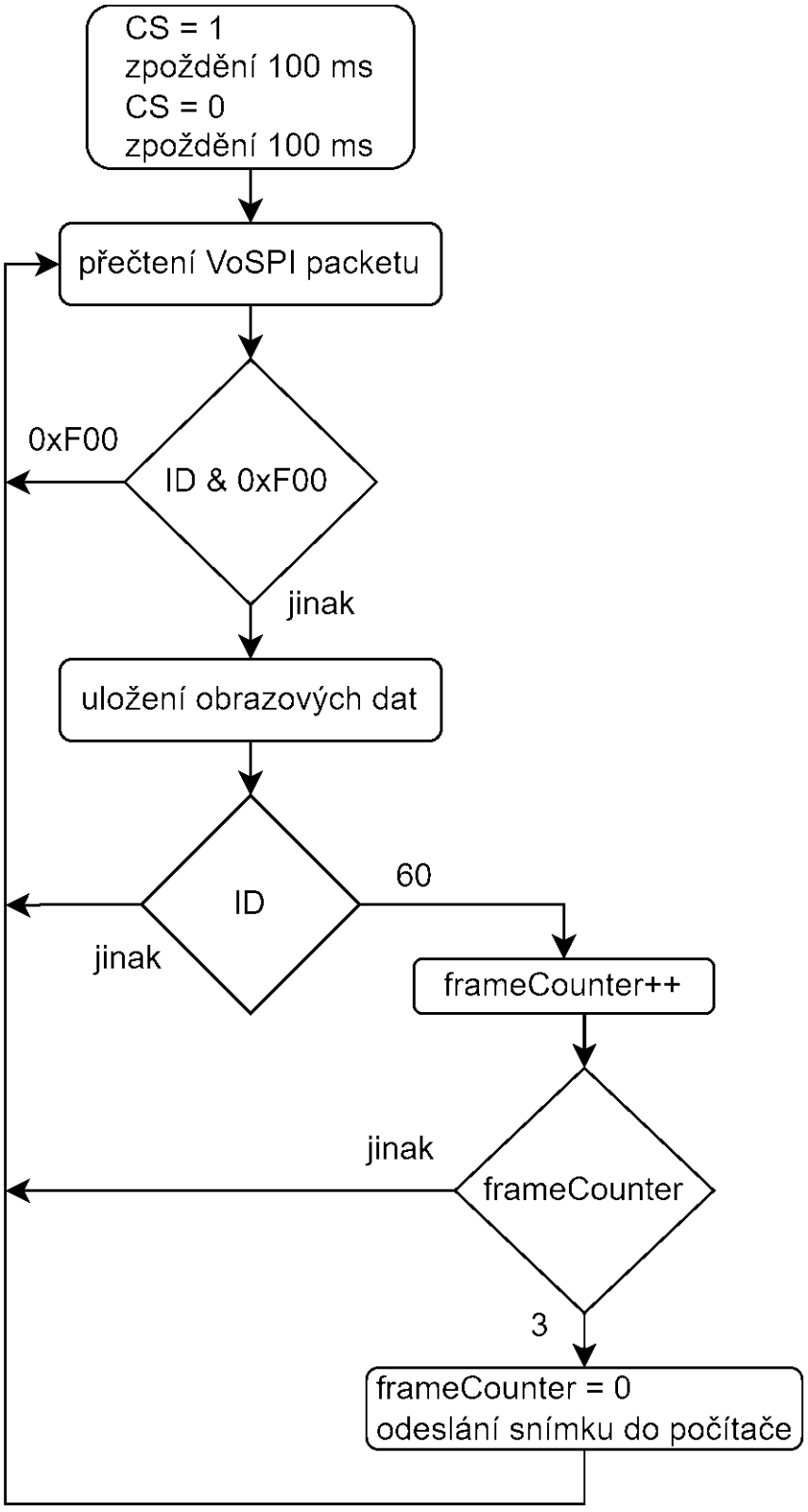
Pro komunikační rozhraní mezi počítačem a STM32 byl vyzkoušen UART a virtuální COM port, z čehož ani jeden způsob nefungoval bezchybně z důvodu velkého objemu přenášených dat na tyto technologie. Výsledky jsou diskutovány v následujících podkapitolách.



Obr. 3.1: Konfigurace kmitočtů sběrnic a periferií.

3.2 Algoritmus čtení dat

Komunikace s kamerou začne vynucením synchronizace, kdy je signál CS nastaven na log. 1, čímž se pozastaví funkce kamery a následným znovunastavením CS na log. 0 začne proces inicializace a synchronizace kamery. Poté nastane samotné čtení VoSPI paketů, pokud se jedná o discard paket, data jsou zahozena a začne čtení nového paketu. Pokud jsou data validní, z paketu jsou získána obrazová data, která se uloží do RAM paměti mikrokontroléru. ID hodnota paketu vyjadřuje pořadí aktuálně přenášeného řádku obrazu, pokud je rovna vertikálnímu rozlišení kamery, v případě této práce 60, označí se momentálně přijímaný snímek za kompletně přijatý. Dále je kontrolováno, zda-li je snímek duplicitní, v tomto případě se snímek dále nezpracovává. Je-li však originální, je odeslán do počítače a čítač originálních snímků je vynulován. Vývojový diagram je uveden na obrázku 3.2.



Obr. 3.2: Vývojový diagram algoritmu čtení dat z kamerového modulu.

3.3 Odesílání přečtených dat

Během odesílání dat do počítače je nutno zajistit, aby prodleva způsobená touto operací nezapříčinila nestihnutí přečtení následujícího snímku z kamery, jinak dojde k desynchronizaci (Obr. 2.6).

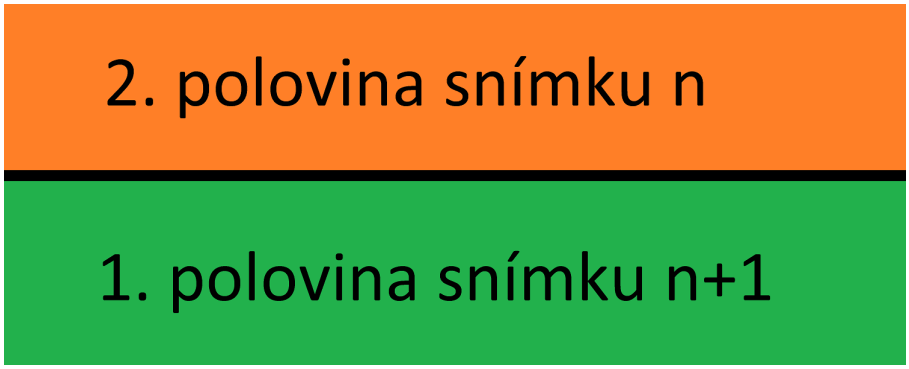
Jeden pixel tvoří 2 byty, aby ho bylo možné přenést přes komunikační rozhraní použitá v této práci, je potřeba ho rozdělit na 2 samostatné byty. Data se poté přenáší v binární formě.

Protože samotné pixely jsou velké pouze 14 bitů a zbylé 2 bity jsou vždy log. 1, byly vyzkoušeny metody pro snížení datového toku, kdy se tyto 2 nepoužité bity nahradí užitečnými bity následujícího snímku. Ukázalo se ale, že prodleva způsobená operacemi pro přerovnání dat je několikanásobně vyšší, než je přijatelné zpoždění, při kterém se stihne přečíst následující snímek z kamery.

3.3.1 Synchronizace s počítačem

Dalším bodem v komunikaci, kde je potřeba synchronizace, je mezi STM32 a počítačem. Protože obrazová data neobsahují žádné dodatečné informace o pozici pixelu, může nastat, že program běžící v počítači začne zpracovávat data od jiného pixelu, než je první v obrazu a nastane splynutí dvou snímků do sebe. Obr. 3.3 ilustruje situaci, kdy program začne zpracovávat datový tok v polovině odesílaného snímku.

Této situaci je možné předejít, když se před samotný začátek odesílání snímku vloží sekvence bitů, která nemůže běžně nastat v datové reprezentaci obrazových bodů a bude unikátně vyjadřovat začátek přenosu tohoto snímku. Protože z 16 přenesených bitů tvoří obrazová data pouze 14 a zbylé 2 jsou vždy na úrovni log. 1, můžeme tyto bity využít pro účel synchronizace, a to s hodnotou log. 0. Přesněji jsou pro synchronizaci použity 2 po sobě jdoucí byty s hodnotou 0x00.



Obr. 3.3: Desynchronizace obrazu při chybném určení nultého pixelu způsobující spojení dvou rozdílných snímků do jednoho.

3.3.2 UART

Přenos dat přes UART je jedna z nejjednodušších možností, jak odesílat obraz z mikrokontroléru do počítače. Použitý STM32 mikrokontrolér obsahuje hned několik periferií rozhraní UART, ze kterých si lze vybrat. Hlavním limitujícím faktorem je přenosová rychlost, resp. baud rate. Potřebný počet bitů pro přenesení za 1 sekundu je vypočítán v rovnici 3.1:

$$n_{bity} = 80 \cdot 60 \cdot 16 \cdot 9 = 691200 \quad [b/s] \tag{3.1}$$

pro rozlišení 80 x 60 o barevné hloubce 16 bitů s přenosem 9 snímků za sekundu. Výsledná hodnota zahrnuje pouze počet bitů obrazových dat, baud rate by musel být ještě vyšší, aby byly v komunikaci zahrnuty i synchronizační byty (podkapitola 3.3.1) a řídící signály protokolu UART.

Během vývoje bylo zjištěno, že při vyšších hodnotách baud rate se komunikace stává nestabilní nebo selže úplně. Nejvyšší stabilní dosažený počet přenesených snímků za sekundu byl přibližně poloviční než nominální, tj. okolo 4 snímků za sekundu. UART periferie samotného mikrokontroléru by měla být schopna dosáhnout mnohem vyšších přenosových rychlostí (v řádu jednotek Mb/s [5]), chyby přenosu byly nejspíše způsobeny použitým převodníkem UART/USB nebo jeho ovladači v systému Windows. Po dalším testování přenosu obrazových dat přes UART by mohlo být dosaženo optimálního výsledku, nicméně pro účely této práce byla tato metoda přenosu dat opuštěna.

3.3.3 Virtuální USB COM port

Použitý STM32 mikrokontrolér obsahuje periferii USB zařízení. S pomocí knihoven dodaných v SDK od výrobce se určí typ zařízení, pod kterým se bude mikrokontrolér jevit hostitelskému počítači. Na výběr je z:

- Audio Device Class
- Communication Device Class (Virtual Port Com)
- Download Firmware Update Class (DFU)
- Human Interface Device Class (HID)
- Custom Human Interface Device Class (HID)
- Mass Storage Class

Virtual Com Port má nejblíže k UART komunikaci, hostitelskému počítači se jeví úplně stejně, ale na straně mikrokontroléru je nutné provést několik malých změn v kódu. Po nahrazení obsluhy UART periferie kódem pro práci s virtuálním COM portem je možné dosáhnout dostatečně vysoké rychlosti na přenos nominálního počtu snímků za sekundu, které kamerový modul vytváří. Pro správnou funkci je vyžadován ovladač virtuálního COM portu od výrobce, který zajišťuje komunikaci mezi mikrokontrolérem a

hostitelským počítačem. Na straně mikrokontroléru se již nepracuje s hodnotou baud rate, mikrokontrolér data odesílá maximální rychlostí. Na straně hostitelského počítače je možné zadat v obslužném programu libovolnou hodnotu baud rate, ovladač ji nicméně ignoruje a přenos dat řeší vlastním způsobem.

Během testování na několika různých počítačích bylo zjištěno, že virtuální COM port nefunguje na každém počítači stejně. Na jednom počítači probíhala komunikace v pořádku, avšak na druhém počítači byl přenos dat opět nestabilní a spojení se ztrácelo. Způsobeno je to s největší pravděpodobností ovladačem pro virtuální COM port, který na odlišných verzí operačního systému a jiném hardwaru počítače nefunguje stejně. Pro stabilizaci datového přenosu na druhém počítači bylo nutno uměle omezit přenosovou rychlost přibližně na polovinu, a i tak se komunikace místy přerušovala.

3.3.4 Další možná rozhraní pro komunikaci

Přenos dat do počítače není pevně závislý na určitém typu komunikačního rozhraní. Pokud jsou splněny podmínky přenosu, aby nedošlo k desynchronizaci s kamerovým modulem, může být použito libovolné rozhraní.

Ethernet se jeví jako slibné rozhraní, zvlášť u mikrokontrolérů s přímou integrací Ethernetové periferie. Technologie podporuje vysoké přenosové rychlosti a odpadly by veškeré problémy s nefunkčními ovladači pro raritní přenosová rozhraní, která nepočítají s relativně vysokým datovým tokem. Další výhodou by byla možnost připojit více kamerových modulů do sítě a centrálně snímat data z několika lokací.

3.4 Popis kódu

3.4.1 Inicializace STM32

Před samotnou inicializací Lepton modulu je provedena inicializace mikrokontroléru a všech použitých periferií, tj. zdroje hodinových signálů, GPIO, SPI a USB pro virtuální COM port.

3.4.2 Popis globálních proměnných

Knihovna obsahuje následující definice polí a proměnných pro práci s kamerovým modulem.

```
1 static uint8_t imageFrame[LEPTON_FRAME_SIZE];
2 static uint8_t image[(IMAGE_LINES*(LEPTON_FRAME_SIZE - 4)) + 2];
3 static uint8_t sentImages = 3;
4 static uint16_t failedFrames = 0;
```

Pole bytových hodnot *imageFrame* slouží k uložení VoSPI paketu, jeho velikost je standardně 164 bytů (podkapitola 2.5.1).

Druhé pole *image* slouží ke sloučení a přípravě přijímaných dat z kamerového modulu. Velikost se odvíjí od rozlišení kamery a přídavných synchronizačních hodnot, vypočte se jako počet horizontálních řádků vynásobených velikostí jednoho tohoto řádku, což je velikost VoSPI paketu zmenšená o první 4 byty, které obsahují ID a CRC (podkapitola 2.5.1). Dále jsou k celkové velikosti přičteny 2 byty, sloužící jako konstanta pro synchronizaci každého následného odeslání snímku do počítače (podkapitola 3.3.1). Výsledná velikost pro účely této bakalářské práce je 9602 bytů.

Proměnná *sentImages* představuje počítadlo odeslaných snímků pro zmenšení výsledného datového toku komunikace s počítačem, kdy se odesílá pouze každý třetí snímek vlivem duplicity (podkapitola 2.5.2). Inicializována je na hodnotu 3, aby byl první přečtený snímek odeslán ihned.

Zbývající proměnná *failedFrames* je použita k počítání sousledně přijatých discard paketů, na jejímž základě je následně vytvořena kontrolní reinicializace spojení s Lepton modulem, která nastane, když z modulu po určitou dobu nepřijde validní paket.

3.4.3 Inicializace Lepton modulu

Inicializace Lepton modulu a spuštění procesu snímání a odesílání dat nastává voláním funkce *LeptonInit*, které je předána reference na handler inicializovaného SPI rozhraní.

```
1 int main(void)
2 {
3           ...
4
5     LeptonInit(hspi1);
6     while (1)
7     {
8     }
9 }
```

Funkce *LeptonInit* inicializuje vybrané globální proměnné, poté inicializuje Lepton modul a nakonec spustí proces čtení a odesílání snímků.

```
1 void LeptonInit(SPI_HandleTypeDef spiHandler)
2 {
3     image[0] = 0;
4     image[1] = 0;
5     spi = spiHandler;
6     InitModule();
7     Start();
8 }
```

Nastavení prvních dvou bytů pole *image* představuje konstantní synchronizační byty komunikace s počítačem.

Funkce *InitModule* zajišťující inicializaci Lepton modulu spočívá v přivedením pulzu na pin CS kamerového modulu s periodou $T = 200\ ms$.

```
1  static void InitModule()
2  {
3      HAL_GPIO_WritePin(GPIOA, GPIO_PIN_4, GPIO_PIN_SET);
4      HAL_Delay(100);
5      HAL_GPIO_WritePin(GPIOA, GPIO_PIN_4, GPIO_PIN_RESET);
6      HAL_Delay(100);
7  }
```

Tato funkce je pak dále využívána k reinicializaci spojení s modulem v případě detekce vysokého počtu discard paketů.

3.4.4 Čtení a odesílání snímků

Proces čtení a odesílání je spuštěn zavoláním funkce *Start*.

```
1  void Start()
2  {
3      while (1)
4      {
5          CheckFailedConnection();
6          HAL_SPI_Receive(&spi, (uint8_t *)&imageFrame[0], 4, 5);
7
8          if (((imageFrame[0] & 0xf) == 0x0f))
9          {
10             DiscardFrame();
11             continue;
12         }
13
14         ReceiveImageData();
15     }
16 }
```

Ve smyčce je periodicky prováděna kontrola spojení, přijmutí záhlaví paketu, vyhodnocení paketu a zpracování zbylých dat paketu.

Jako první je kontrolováno spojení na základně počtu přijatých discard paketů ve funkci *CheckFailedConnection*.

```
1  static void CheckFailedConnection()
2  {
3      if (failedFrames > 5000)
4      {
5          InitModule();
6          failedFrames = 0;
7      }
8  }
```

V případě, že počet sousledně přijatých discard paketů převyšuje stanovenou konstantu, je vyvolána reinicializace spojení a počítadlo *failedFrames* se vynuluje.

Následně jsou přijaty první 4 byty (záhlaví) VoSPI paketu do pole *imageFrame* a vyhodnocuje se, zdali je paket validní, nebo se jedná o discard paket. V případě, že

přijímaný paket není validní, neobsahuje tedy obrazová data, stále je ho nutno celý přečíst. To zajišťuje zavolání funkce *DiscardFrame*, jenž dále volá funkci *SkipSpiImageData*.

```
1  static void DiscardFrame()
2  {
3      failedFrames++;
4      SkipSpiImageData();
5  }
6
7  static void SkipSpiImageData()
8  {
9      HAL_SPI_Receive(&spi, (uint8_t *)&imageFrame, (LEPTON_FRAME_SIZE - 4)
           , 5);
10 }
```

Dojde ke zvýšení počítadla discard paketů *failedFrames* a je přečteno zbylých 160 bytů VoSPI paketu, které jsou uloženy do zbývající části pole *imageFrame*, která v tomto případě slouží pouze jako způsob zahození přicházejících dat z Lepton modulu. Nebyl zjištěn jiný způsob, jak pomocí HAL knihovny přečíst data z SPI bez ukládání do paměti, což zbytečně zvyšuje potřebnou velikost paměti dat mikrokontroléru. Použitý mikrokontrolér v této bakalářské práci má však dostatečně velkou paměť dat a tento způsob zahození dat nezpůsobil žádné problémy. Druhý způsob, který by nepotřeboval dodatečnou paměť, by byl přímo přečíst data z SPI bez HAL knihovny pomocí CMSIS.

V druhém případě, kdy je paket validní a obsahuje obrazová data, je volána funkce *ReceiveImageData*.

```
1  static void ReceiveImageData()
2  {
3      uint8_t frameNumber = imageFrame[1];
4
5      if (frameNumber >= IMAGE_LINES)
6      {
7          SkipSpiImageData();
8          return;
9      }
10
11     failedFrames = 0;
12     HAL_SPI_Receive(&spi, (uint8_t *)&image[2 + (frameNumber*(
           LEPTON_FRAME_SIZE - 4))], (LEPTON_FRAME_SIZE - 4), 5);
13     if (IsImageComplete(frameNumber))
14     {
15         ImageReceived(frameNumber);
16     }
17 }
```

Nejprve je načteno do proměnné *frameNumber* ID z přečteného záhlaví VoSPI paketu. Přestože je ID 2bytové, užitečná hodnota je vždy v méně významném bytu, protože ID reprezentuje pořadí horizontálního řádku snímku, které nikdy nepřesáhne rozsah jednoho

bytu. Poté je kontrolováno pořadí řádku s konstantou *IMAGE_LINES*, přes kterou lze uměle omezit počet přijímaných řádků a přijímat například pouze polovinu obrazu. Tato funkce byla využita zejména při vývoji a nyní slouží pouze jako dodatečná kontrola validity dat.

V případě, kdy jsou data označena za validní, počítadlo discard paketů *failedFrames* je vynulováno a obrazová data jsou přijata z kamerového modulu přes SPI do pole *image*. Je potřeba správně určit počáteční pozici dat v paměti, aby nedošlo k přepsání předchozích přijatých obrazových dat. Pozice se vypočítá jako velikost předchozích dat, které je nutno přeskočit, neboli vynásobení pořadí aktuálně přijímaného řádku s velikostí jednoho řádku. K pozici je dodatečně přidán offset 2 bytů, které představují konstantní synchronizační byty na začátku paměti.

Nakonec je testováno pomocí funkce *IsImageComplete*, zdali se už přečetl snímek celý.

```
1 static uint8_t IsImageComplete(uint8_t frameNumber)
2 {
3     return frameNumber == (IMAGE_LINES - 1);
4 }
```

Kontrola spočívá v porovnání pořadí přijatého snímku s definovaným počtem řádků na jeden snímek *IMAGE_LINES* zmenšeným o 1, protože pořadí řádků začíná od 0. V případě, že je snímek kompletní, je zavolána funkce *ImageReceived*.

```
1 static void ImageReceived(uint8_t frameNumber)
2 {
3     if (sentImages == 3)
4     {
5         sentImages = 0;
6         SendImage();
7     }
8     sentImages++;
9 }
```

Funkce obaluje samotnou funkci *SendImage*, sloužící pro finální odeslání snímku do počítače, kontrolou duplicity snímků pro zmenšení datového toku.

Konečné odeslání snímku přes virtuální COM port zajišťuje funkce obsluhy USB periferie *CDC_Transmit_FS*, které je předána adresa počátku dat a počet bytů k odeslání. Vrací USBD status reprezentující stav odeslání dat. Pomocí while smyčky se čeká, dokud vrácený stav není *USBD_OK* značící úspěšné přenesení dat.

```
1 static void SendImage()
2 {
3     while(CDC_Transmit_FS(&image[0], ((LEPTON_FRAME_SIZE - 4) *
          IMAGE_LINES)+2) != USBD_OK);
4 }
```

3.5 Úprava kódu pro jiné typy mikrokontrolérů

Při vývoji byla použita HAL knihovna, do určité míry by mělo být možné použít téměř nezměněný kód pro mikrokontroléry z rodiny STM32. Pro ostatní typy mikrokontrolérů bude nutné upravit části kódu, kde probíhá SPI komunikace s kamerovým modulem a komunikace s počítačem přes virtuální COM port. Samotná část s algoritmem pro čtení a vyhodnocování dat z kamery nevyužívá žádné specifické vlastnosti použitého mikrokontroléru, měla by se tedy také obejít bez nutnosti zásadních úprav.

4

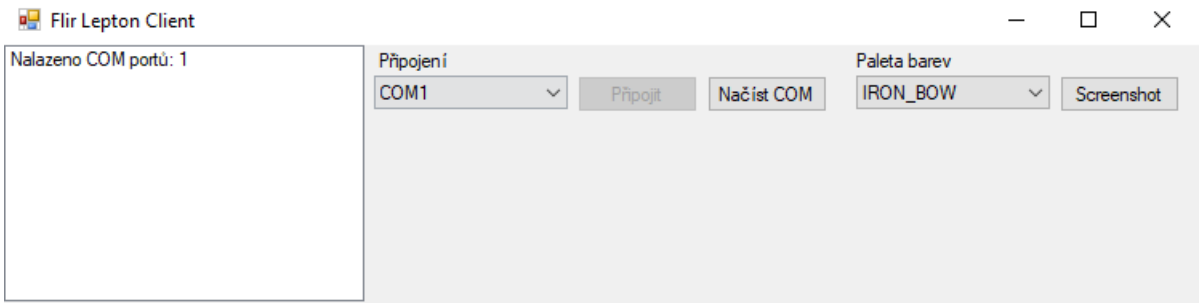
Zpracování pořízených snímků

4.1 Použité technologie

Aplikace zajišťující příjem, zpracování a grafické zobrazení dat z Lepton modulu je vytvořena v jazyce C# využívající .NET Framework 4.6.1 a Windows Forms pro grafické rozhraní. Aplikace je zejména založena na knihovnách z .NET Frameworku, které obsahují třídu SerialPort pro komunikaci s kamerovým modulem přes sériový port a třídu Graphics pro vykreslování a zaznamenávání obrazu. V případě použití jiné technologie bude nutno nahradit hlavně tyto knihovny jinými, samotný princip čtení dat, synchronizace a práce s daty zůstává však stejný.

Vývoj aplikace proběhl ve vývojovém prostředí Visual Studio a pro verzování kódu byl použit nástroj Git.

4.2 Uživatelské rozhraní aplikace



Obr. 4.1: Uživatelské rozhraní aplikace.

Uživatelské rozhraní tvoří tři hlavní části. V levém panelu je konzole pro textový výstup, ve které je uživatel jednak informován o aktuálním stavu aplikace, jako je počet nalezených COM portů a stav připojení. A také je dále využita pro zobrazení teploty ve zvolených bodech obrazu.

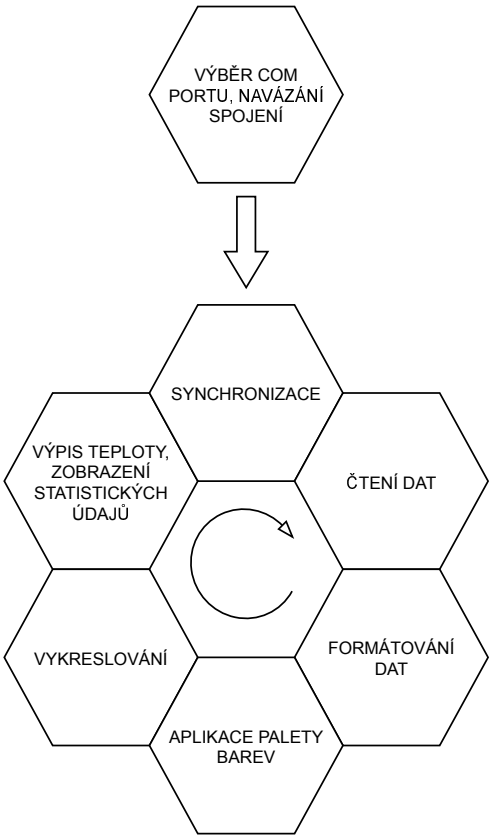
Horní panel obsahuje uživatelské prvky pro načtení a výběr COM portu, ke kterému je připojený mikrokontrolér s kamerovým modulem. Dále je možno za běhu měnit palety barev (podkapitola 4.4.3) a ukládat zobrazovaný snímek do souboru.

Zbylý prostor je využit pro samotné grafické zobrazení přijímaných snímků z kamery, jeho velikost je dynamicky přizpůsobena aktuální velikosti okna aplikace tak, aby byl zachován poměr stran obrazu z kamery. Při kliknutí myší do libovolného bodu v obrazu se vypíše teplota v tomto bodě.

4.3 Čtení a zpracování dat

Po úspěšném navázání komunikace přes sériový port začne proudit datový tok z mikrokontroléru, který je ukládán do vstupního zásobníku knihovny sériového portu. Před přijetím každého nového snímku je spuštěn proces synchronizace.

Poté začne čtení samotných obrazových dat. Probíhá tak, že se čeká, dokud ve vstupním zásobníku není jeden celý VoSPI paket, tj. 160 bytů v případě této práce (podkapitola 2.5.1). Následně jsou přijaté 8bitové hodnoty spojeny do 16bitových, které již reprezentují jeden celý obrazový bod a jsou uloženy do paměti. Pokud je vyhodnoceno kompletní přijetí snímku, data jsou předána ke zpracování a celý proces začíná znovu pro následující snímek.



Obr. 4.2: Cyklus aplikace.

4.3.1 Čtecí smyčka

Do čtecí smyčky je vstoupeno spuštěním metody *ReceiveData*. Metoda je spuštěna v novém vlákně, aby přijímání dat probíhalo paralelně se zbylými částmi aplikace, tj. vykreslování a obsluha uživatelského rozhraní.

```
1  private void ReceiveData()
2  {
3      while (true)
4      {
5          if (isSyncNeeded)
6          {
7              Sync();
8          }
9
10         if (serialPort.BytesToRead >= 160)
11         {
12             byte[] buffer = new byte[160];
13             serialPort.Read(buffer, 0, 160);
14             ProcessRow(buffer);
15         }
16         else
17         {
18             Thread.Sleep(5);
19         }
20     }
21 }
```

Na začátku přijímání každého nového snímku je vyžadována synchronizace, její potřebu signalizuje proměnná *isSyncNeeded*, která je nastavena vždy po přijetí celého snímku a je resetována po úspěšné synchronizaci. Následně je kontrolováno, zdali je ve vstupním zásobníku sériového portu již připraven celý VoSPI paket. Pokud ano, paket je zpracován, v opačném případě je vlákno uspáno na 5 ms pro uvolnění zdrojů procesoru. Měření využití procesoru ukázalo, že uspáním vlákna došlo ke snížení využití jádra procesoru z téměř 100% využití na nízké jednotky procent.

4.3.2 Synchronizace

Synchronizaci zajišťuje metoda *Sync*, po jejím zavolání vstoupí vlákno programu do smyčky, která blokuje zpracování obrazových dat do té doby, dokud není přijata a vyhodnocena definovaná sekvence synchronizačních bytů.

```
1  private void Sync()
2  {
3      bool synchronized = false;
4      while (true)
5      {
6          if (serialPort.BytesToRead > 0)
7          {
8              byte syncByte = (byte)serialPort.ReadByte();
9              if (syncByte == 0 && synchronized)
10             {
11                 isSyncNeeded = false;
12                 break;
13             }
14
15             synchronized = syncByte == 0;
16         }
17         else
18         {
19             Thread.Sleep(5);
20         }
21     }
22 }
```

Synchronizace spočívá v přijetí dvou po sobě jdoucích bytů s nulovou hodnotou, podrobněji tato problematika vysvětlena v podkapitole 3.3.1. Z vstupního zásobníku sériového portu jsou data čtena po jednotlivých bytech. Pokud byl předchozí přečtený byte nula (vyjádřeno proměnnou *synchronized*) a zároveň i aktuální byte je nula, ze synchronizační smyčky je vystoupeno a program je připraven na zpracování obrazových dat, která budou následovat.

Opět je kontrolován počet bytů ve vstupním zásobníku sériové portu, pokud je zásobník prázdný, vlákno je uspáno ze stejných důvodů, jako v předchozí podkapitole.

4.3.3 Zpracování přijatých dat

Metoda *ProcessRow* slouží ke zpracování jednoho VoSPI paketu, který představuje jeden řádek obrazu. Převádí dvě po sobě jdoucí přijaté 8bitové hodnoty na jednu 16bitovou hodnotu, kterou ukládá do proměnné *tempReading*.

```
1  private void ProcessRow(byte[] data)
2  {
3      for (int i = 0; i < leptonCols; i++)
4      {
5          ushort tempReading = (ushort)(data[2 * i] << 8 | data[2 * i + 1])
               ;
6          receivingImage.SetTempReading(currentRow, i, tempReading);
7      }
8      currentRow++;
9
10     if (currentRow == leptonRows)
11     {
12         FrameHasBeenReceived();
13     }
14 }
```

Výsledek v proměnné je uložen do objektu *receivingImage*, který je instancí třídy *LeptonImage*, pomocí metody *SetTempReading*, která převede 16bitovou hodnotu vyjadřující absolutní teplotu v Kelvinech na teplotu ve stupních Celsia pomocí metody *ConvertLeptonValue*. Hodnota je následně uložena na patřičný řádek a sloupec ve výsledném obrazu.

```
1  public class LeptonImage
2  {
3      private double[,] data;
4
5      public LeptonImage(int rows, int cols)
6      {
7          data = new double[rows, cols];
8          ...
9      }
10
11     public void SetTempReading(int row, int column, ushort value)
12     {
13         data[row, column] = ConvertLeptonValue(value);
14     }
15
16     private double ConvertLeptonValue(ushort value)
17     {
18         return Math.Round((value * 0.01) - 273.15, 2);
19     }
20
21     public TempReading GetMaxTemp() { ... }
22     public TempReading GetMinTemp() { ... }
23 }
```

Objekt dále obsahuje metody pro získání extrémů teploty ve snímku, které jsou využity pro kalibraci palety barev, popsané v následujících podkapitolách, a také jsou průběžně zobrazovány uživatelovi v aplikaci ve statistikách.

Splnění podmínky rovnosti proměnných *currentRow* a *letonRows* značí, že byl již celý

snímek přijat a je volána metoda *FrameHasBeenReceived*, která ukončí příjem aktuálního snímku, zpřístupní výsledný objekt *receivingImage* k vyzvednutí další části aplikace a připraví program pro příjem následujícího snímku.

4.4 Vykreslování snímků

Část programu zajišťující vykreslování snímků pracuje pouze již s objektem třídy *LeptonImage*, který byl vytvořen v předchozí kapitole a obsahuje normalizovaná data nezávisející na modelu kamerového modulu.

Před vykreslením každého snímku je vytvořena instance vybrané palety barev, pomocí které je určena barva každého pixelu. Vykreslování zajišťuje třída Graphics, vytváří bitmapový obrázek o stejné velikosti, jako je rozlišení Lepton modulu, tj. pro tuto bakalářskou práci 80 x 60 pixelů. Následně je bitmapa vykreslena na obrazovku do pracovní oblasti aplikace, rozlišení obrazu je automaticky knihovnou přepočítáno tak, aby byla využita co největší plocha obrazovky při zachování poměru stran původního obrazu.

Tento způsob vykreslování, které je provedeno dvakrát, místo přímého vykreslení zdrojových dat přímo na obrazovku v požadovaném rozlišení, má dvě výhody. První výhodou je, že knihovna při zvýšení rozlišení původní bitmapy automaticky aplikuje na obraz vyhlazující filtry, které zmírní přechody mezi jednotlivými pixely. Druhou výhodou je, že lze bitmapu exportovat do souboru a rozlišení výsledného obrázku není závislé na aktuální velikosti okna aplikace. Export obrazu funguje na vyžádání uživatele stiskem tlačítka, případně by bylo možné obraz ukládat souvisle k získání videozáznamu.

4.4.1 Vykreslování do bitmapy

K získání bitmapy slouží třída *LeptonDraw*, která je potomkem třídy *Control* a je použita jako komponenta v uživatelském rozhraní Windows Forms. Každý zpracovaný snímek vyvolá metodu *SetImage*.

```
1  public void SetImage(LeptonImage image)
2  {
3      this.image = image;
4      maxTemp = image.GetMaxTemp();
5      minTemp = image.GetMinTemp();
6      linearColor = new LinearColorFactory().GetLinearColor(ColorPallete,
           minTemp.Temp, maxTemp.Temp);
7
8      DrawBitmap();
9      ...
10 }
```

Zde je vytvořena kalibrovaná instance palety barev *linearColor*, kterou si uživatel zvolil ve vlastnosti *ColorPallete*. Se všemi připravenými daty je poté spuštěn proces vytvoření bitmapy přes metodu *DrawBitmap*.

```
1  private void DrawBitmap()
2  {
3      imageBitmap = new Bitmap(leptonCols, leptonRows);
4      Graphics graphics = Graphics.FromImage(imageBitmap);
5      for (int col = 0; col < leptonCols; col++)
6      {
7          for (int row = 0; row < leptonRows; row++)
8          {
9              double temp = image.GetTempReading(row, col);
10             DrawPixel(graphics, row, col, temp);
11         }
12     }
13 }
```

Nejprve je vytvořen objekt bitmapy o velikosti rozlišení Lepton modulu. Z této bitmapy je následně vytvořen objekt třídy *Graphics*, který do ní umožňuje vykreslovat základní tvary. Dva vnořené cykly prochází a vykreslují jednotlivé obrazové body metodou *DrawPixel*.

```
1  private void DrawPixel(Graphics graphics, int row, int col, double value)
2  {
3      Color color = linearColor.GetColor(value);
4      SolidBrush brush = new SolidBrush(color);
5      graphics.FillRectangle(brush, col, row, 1, 1);
6  }
```

Tato metoda zabaluje získání barvy aktuálního pixelu přes instanci palety barev *linearColor* a následné vykreslení třídou *Graphics*. Vykreslení samotného pixelu je dosaženo metodou *FillRectagle* s parametry pro vykreslení čtverce o velikosti 1 x 1 pixelu.

Tímto je proces vytvoření bitmapy dokončen a její objekt je připraven k vykreslení na obrazovku nebo k uložení obrázku do souboru. Uložení do souboru lze vyvolat tlačítkem v aplikaci, je pro to využita metoda objektu bitmapy *Save*, která uloží obraz do souboru pod zvoleným názvem ve vybraném formátu. Pro tuto práci byl zvolen formát PNG.

4.4.2 Vykreslování na obrazovku

Vykreslování probíhá v překryté metodě *OnPaint*, zděděné ze třídy *Control*. Volání probíhá buď automaticky frameworkem Windows Forms, a to při změně velikosti okna a podobných událostech, nebo je vynuceno metodou *Invalidate* pokaždé, když je dostupný nový snímek k vykreslení.

```
1  protected override void OnPaint(PaintEventArgs e)
2  {
3          ...
4      e.Graphics.DrawImage(imageBitmap, 0, 0, (float)PixelSize * leptonCols
           , (float)PixelSize * leptonRows);
5          DrawInfo(e.Graphics);
6  }
```

Při vykreslování musí být zajištěno, aby byl zachován poměr stran. Je proto spočítána optimální velikost jednoho pixelu tak, aby byl poměr stran zachován a zároveň aby byla využita pracovní plocha aplikace co nejvíce. Velikost pixelu vrací vlastnost *PixelSize* dynamicky v závislosti na aktuálním rozlišením okna aplikace.

```
1  private double PixelSize
2  {
3      get
4      {
5          double widthSize = Width / (double)leptonCols;
6          double heightSize = Height / (double)leptonRows;
7
8          return widthSize > heightSize ? heightSize : widthSize;
9      }
10 }
```

Určení velikosti spočívá v odděleném určení velikost pixelu pro řádek a sloupec, následně jsou velikosti porovnány a použita je ta nižší hodnota. Tímto způsobem, pokud poměr stran okna aplikace neodpovídá poměru stran bitmapy, bude okno využito naplno alespoň buď horizontálně, nebo vertikálně.

Metoda *DrawInfo* slouží k vykreslení statistik ohledně teploty a počtu přijímaných snímků za sekundu do horního levého rohu obrazu. Vykreslení probíhá pouze na obrazovku, při ukládání bitmapy do souboru není zahrnuto.

```
1  private void DrawInfo(Graphics graphics)
2  {
3      Font font = new Font("Consolas", 8);
4      if (fpsSw.ElapsedMilliseconds >= 2000)
5      {
6          fps = fpsCounter / (fpsSw.ElapsedMilliseconds / (double)1000);
7          fps = Math.Round(fps, 2);
8          fpsCounter = 0;
9          fpsSw.Restart();
10     }
11
12     string info = String.Format("FPS:      {0}\r\nMin temp: {1} °C\r\nMax
           temp: {2} °C", fps, minTemp.Temp, maxTemp.Temp);
13     graphics.DrawString(info, font, Brushes.White, 0, 0);
14 }
```

Extrémy teplot aktuálního snímku jsou získány z objektu obrázku třídy *LeptonImage* a jsou přímo zobrazeny. Počet snímků za sekundu je průběžně získáván z počítadla snímků, které je každé 2 sekundy přepočteno, vypsáno obrazovku a následně vynulováno. Font použitý pro vykreslování statistik má pevnou šířku písma, byl zvolen z toho důvodu, aby bylo možné jednoduše text formátovat jako tabulku.

4.4.3 Palety barev

Data získaná z termální kamery představují teplotu jednotlivých obrazových bodů, tyto hodnoty lze chápat jako odstíny šedi a přímo je vykreslit. Problém je, že rozsah teplot kamery je ve stovkách stupních Celsia. Pokud by se tedy bez jakéhokoliv dalšího přepočtu pevně přiřadily odstíny šedi na celý teplotní rozsah a snímané objekty kamerou by měly například teplotní rozdíl pouze 10 °C, tento rozdíl by byl téměř nepozorovatelný.

Pro lepší grafickou reprezentaci dat jsou použity palety barev, které řeší tento problém tak, že extrémy odstínů šedi (bílou a černou barvu) přiřadí na extrémy teploty (minimální a maximální teplotu) a hodnoty mezi nimi lineárně rozloží. Tento proces je dále označován jako normalizace. Ve výsledném obrazu je poté mnohem jednodušší rozpoznat rozdíly teplot objektů. Normalizaci je nutno provést pro každý nový snímek, ve kterém se vyskytují jiné extrémy teplot.

Druhou funkci, kterou paleta barev plní, je možnost místo odstínů šedi použít odstíny libovolných barev, výsledná barva je poté lineární gradient extrémů zvolených dvou barev pro aktuální teplotu bodu. Pro tuto bakalářskou práci byla vyvinuta třída palet barev, ve které je možné nadefinovat libovolný počet těchto přechodů. Ke každé barvě je přiřazena hranice vyjádřená jako procentuální šířka normalizovaného obrazu. To znamená, že například přechod mezi černou, červenou a modrou barvou by byl definován následovně:

- 0 % - černá
- 50 % - červená
- 100 % - modrá

4.4.4 Implementace palet barev

Palety barev jsou definovány polem objektů *ColorRange* (dále jen rozsah), který obsahuje počáteční a koncovou barvu, která je lineárně přiřazena k počáteční a koncové teplotě.

```
1 public class ColorRange
2 {
3     public Color MinColor { get; private set; }
4     public Color MaxColor { get; private set; }
5     public double MinValue { get; private set; }
6     public double MaxValue { get; private set; }
7
8     ...
9 }
```

S polem rozsahů pracuje objekt třídy *LinearColor*, ve kterém probíhá samotný výpočet výsledné barvy metodou *GetColor*.

```
1  public Color GetColor(double value)
2  {
3      ColorRange range = colorRanges.FirstOrDefault(x => x.MinValue <=
           value && x.MaxValue >= value);
4      if (range == null)
5      {
6          throw new LinearColorException();
7      }
8
9      return GetGradientColor(range, value);
10 }
```

Nejprve je nalezen patřičný rozsah, do jehož intervalu minimální a maximální teploty spadá teplota, pro kterou se zjišťuje barva. Následně v zavolané metodě *GetGradientColor* proběhne výpočet jednotlivých RGB složek barvy, ze kterých je složena výsledná struktura barvy *Color*.

```
1  private Color GetGradientColor(ColorRange colorRange, double value)
2  {
3      int r = GetLinearColor(colorRange.MinColor.R, colorRange.MaxColor.R,
           colorRange.MinValue, colorRange.MaxValue, value);
4      int g = GetLinearColor(colorRange.MinColor.G, colorRange.MaxColor.G,
           colorRange.MinValue, colorRange.MaxValue, value);
5      int b = GetLinearColor(colorRange.MinColor.B, colorRange.MaxColor.B,
           colorRange.MinValue, colorRange.MaxValue, value);
6
7      return Color.FromArgb(r, g, b);
8  }
```

Výpočet RGB složek probíhá v metodě *GetLinearColor*, ve které je výsledná hodnota složky barvy vypočtena lineární regresí.

```
1  private int GetLinearColor(int from, int to, double minValue, double
       maxValue, double value)
2  {
3      double a = (to - from) / ((double)maxValue - minValue);
4      double b = from - (a * minValue);
5      return (int)(a * value + b);
6  }
```

Proměnné a a b představují koeficienty přímky:

$$y = ax + b \tag{4.1}$$

4.4.5 Definice palet barev

Palety barev jsou definovány objektem třídy *LinearColorBuilder*, který vrací kalibrovaný objekt třídy *LinearColor* pro aktuální snímek. Složky barev jsou přidány metodou *AddColorRange*, které je předána konečná barva gradientu *color* a desetinné číslo *percentage*, vyjadřující procentuální hranici, po kterou platí tento rozsah.

```
1 public LinearColorBuilder AddColorRange(Color color, double percentage)
2 {
3     double newMaxValue = (DeltaValue * percentage) + lastMinValue;
4     ColorRange colorRange = new ColorRange(lastMinColor, color,
          lastMinValue, newMaxValue);
5     colorRanges.Add(colorRange);
6     lastMinValue = newMaxValue;
7     lastMinColor = color;
8     return this;
9 }
```

Vlastnost *DeltaValue* je rozdíl extrémů teplot zastoupených v aktuálním snímku.

Po dokončení přidávání libovolného množství složek barev je zavolána metoda *Make*, která vrací výsledný objekt třídy *LinearColor*, který je nyní připraven k použití.

```
1 public LinearColor Make()
2 {
3     return new LinearColor(colorRanges);
4 }
```

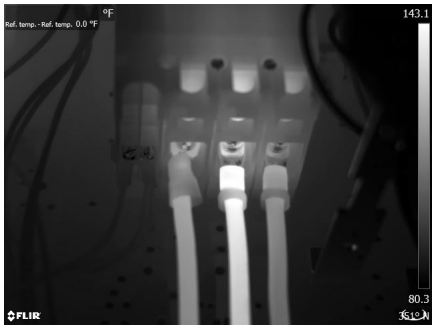
4.4.6 Definice některých palet barev

Pro tuto práci byly připraveny 4 palety barev, a to:

- Greyscale
- Ironbow
- Rainbow
- Outdoor alert

Definice byly převzaty z [6] a [7].

Greyscale (Obr. 4.3) je nejzákladnější paleta barev, která se skládá pouze z odstínů šedi, kde černá barva představuje studené objekty a bílá barva teplé objekty. Existuje také inverzní verze, která má opačnou tepelnou závislost.

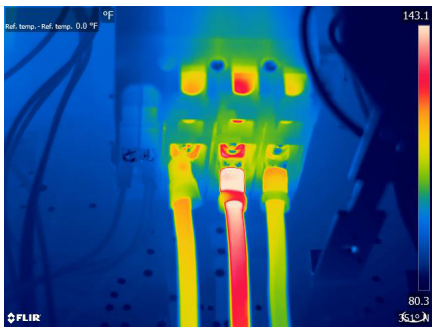


Obr. 4.3: Greyscale paleta barev |Převzato z [6]|

Ironbow paleta (Obr. 4.4) slouží k obecnému použití, tmavé barvy vyjadřují studené objekty a světlé barvy teplé objekty.

Obr. 4.4: Ironbow paleta barev |Převzato z [6]|

Rainbow paleta (Obr. 4.4) je podobná Ironbow paletě, obsahuje ale více barevných složek, které umožňují přesnější rozpoznání menších rozdílů v teplotách.



Obr. 4.5: Rainbow paleta barev |Převzato z [6]|

Outdoor alert (Obr. 4.6) je speciální verze inverzní palety Greyscale, kde 10 % nejvyšších teplot snímku tvoří přechod žluté a oranžové barvy. Paleta je vhodná pro rozpoznání lidí a zvířat ve venkovních prostorech.

Obr. 4.6: Outdoor alert paleta barev |Převzato z [7]|

Instanci třídy *LinearColor* ve zvolené paletě barev vytváří objekt *LinearColorFactory*. Například paleta barev Ironbow je definovaná v kódu následovně:

```
1  private LinearColor GetIronBow(double minValue, double maxValue)
2  {
3      LinearColorBuilder builder = new LinearColorBuilder(minValue,
           maxValue, Color.Black);
4      builder.AddColorRange(Color.FromArgb(9, 0, 110), 1 / (double)17);
5      builder.AddColorRange(Color.FromArgb(180, 0, 150), 4 / (double)17);
6      builder.AddColorRange(Color.FromArgb(236, 87, 0), 4 / (double)17);
7      builder.AddColorRange(Color.White);
8      return builder.Make();
9  }
```

Přes objekt třídy *LinearColorBuilder* jsou postupně vytvořeny rozsahy barev, začínající od černé barvy, přes jednotlivé barevné složky a končící bílou barvou.

Pořízené snímky v rámci této bakalářské práce jsou v příloze A.

4.4.7 Výpis teploty ve zvoleném bodě

Zdrojový objekt teplot třídy *LeptonImage* je po celou dobu vykreslování dostupný v paměti, je tedy možné kdykoliv přistoupit k jednotlivým teplotám pixelů. Toho je využito pro funkci výpisu teploty ve zvoleném bodě, kdy je sledována událost kliknutí myši na zobrazovací plochu aplikace pro snímky.

```
1  protected override void OnMouseClick(MouseEventArgs e)
2  {
3      int row = (int)(e.Y / PixelSize);
4      int col = (int)(e.X / PixelSize);
5
6      double temp = image.GetTempReading(row, col);
7      log.LogMessage($"t({row}, {col}) = {temp} °C");
8  }
```

Získané souřadnice kliknutí z objektu *e* třídy *MouseEventArgs* vyjadřují pozici pixelu vykreslovací plochy, přičemž levý horní roh je počátek souřadnicového systému. Jelikož velikost vykresleného pixelu z termálního snímku neodpovídá velikosti pixelu aplikace, je nutné provést přepočet. Výsledná souřadnice je získána vydělením aktuální velikostí pixelu snímku *PixelSize* a následným zaokrouhlením desetinného místa směrem dolů.

4.5 Možné úpravy aplikace

Jádro aplikace, které zajišťuje připojení, synchronizaci, přijímání dat a následné formátování, je možné využít jako knihovnu pro jinou aplikaci. Třídu pro definici palet barev a výpočet gradientu je možné rozšířit o další palety barev, případně celou třídu využít v jiné aplikaci. Předpokladem je použití stejných technologií.

Vykreslování obrazu, zajištěné třídou Graphics, nepodporuje hardwarovou akceleraci, tj. vykreslování nevyužívá grafickou kartu. Pokud by bylo třeba zpracovávat větší počet grafických dat (vyšší rozlišení nebo snímkovou frekvenci) a nastal by problém, že vykreslování využívá příliš vysoké zdroje procesoru, lze knihovnu nahradit alternativní knihovnou s podporou hardwarové akcelerace, jako je například OpenGL.

5

Závěr

Cílem této bakalářské práce bylo prozkoumat použití modulů termálních kamer Lepton od firmy FLIR a vytvořit ukázkové řešení použití. Přesněji bylo zadáno vytvořit program běžící na mikrokontroléru řady STM32, který naváže komunikaci s modulem termální kamery a bude z něho číst obrazová data, která následně přes USB rozhraní odešle do počítače. Druhá část zadání byla vytvořit aplikaci běžící na počítači, která odeslaná data z mikrokontroléru přijme, zpracuje a zprostředkuje jejich grafické zobrazení.

První část zadání byla splněna a byl vyvinut program pro mikrokontrolér STM32, pro vývoj byl použit vývojový modul Blue Pill. Bylo prozkoumáno několik různých způsobů řešení jednotlivých částí algoritmu, od čtení dat z kamerového modulu, přes zpracování těchto dat až po odeslání dat do počítače. Každou část bylo nutno optimalizovat z hlediska časové náročnosti běhu programu, aby byly dodrženy podmínky na časování použitých technologií.

Výsledkem je řešení, které komunikuje s kamerovým modulem přes SPI rozhraní a odesílá snímky do počítače přes USB virtuální COM port. Bylo vyzkoušeno více komunikačních rozhraní mezi mikrokontrolérem a počítačem, avšak žádné se neukázalo jako dostatečně spolehlivé pro využití maximálního potenciálu kamery. Bylo zjištěno, že spolehlivost přenosu dat je závislá na verzi operačního systému, verzi ovladačů a hardwaru počítače. V práci byly podrobně stanoveny podmínky na komunikační rozhraní, aby bylo možné program upravit a implementovat vlastní přenosové rozhraní, které bude splňovat potřebnou spolehlivost přenosu pro dané použití.

Druhá část zadání, týkající se vytvoření aplikace běžící na počítači a zpracovávající data z mikrokontroléru, byla také splněna. Pro vývoj aplikace byl zvolen programovací jazyk C# s technologií .NET Framework. Aplikace byla vytvořena modulárním způsobem, kdy jsou jednotlivé části od sebe odděleny a díky tomu je možné části aplikace nezávisle na sobě podle potřeby upravovat a používat. První část tvoří komunikační rozhraní s mikrokontrolérem, které udržuje spojení a přijímá data, která následně zpracovává do normalizovaného objektu termálního snímku. Tyto data jsou následně předána do grafické části aplikace, která na ně aplikuje palety barev a vytvoří barevnou reprezentaci snímku. Poslední část tvoří vykreslování snímku na obrazovku, kde je umožněno zobrazit přesnou

hodnotu teploty ve zvoleném bodě.

Pro práci s paletami barev byl vytvořen samostatný systém, přes který lze jednoduše definovat, spravovat a přepínat několik palet barev. Grafický výstup aplikace je tedy možné zásadně ovlivnit pro specifické účely použití.

Literatura

[1] FLIR. *Module Lepton* [online]. [Cit. 27.03.2022]. Dostupné z: https://flir.eu/products/lepton/

[2] FLIR. *The FLIR Lepton* [online]. [Cit. 27.03.2022]. Dostupné z: https://lepton.flir.com/LWIR Micro Thermal Camera

[3] SparkFun Electronics. *FLiR Dev Kit - KIT-13233* [online]. [Cit. 27.03.2022]. Dostupné z: https://www.sparkfun.com/products/retired/13233

[4] FLIR. *FLIR LEPTON® Engineering Datasheet* [online]. [Cit. 08.04.2022]. Dostupné z: https://www.flir.com/globalassets/imported-assets/document/flir-lepton-engineering-datasheet.pdf

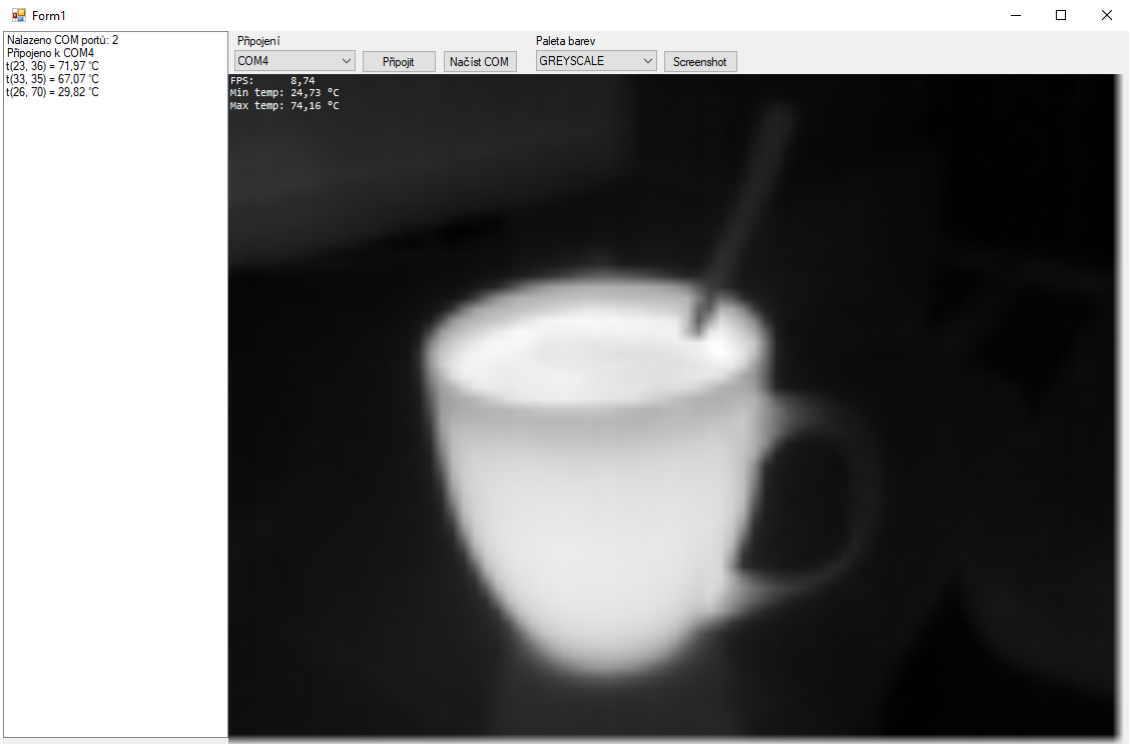
[5] STMicroelectronics. *Medium-density performance line Arm®-based 32-bit MCU with 64 or 128 KB Flash, USB, CAN, 7 timers, 2 ADCs, 9 com. interfaces* [online]. [Cit. 10.04.2022]. Dostupné z: https://www.st.com/resource/en/datasheet/stm32f103c8.pdf

[6] FLIR. *Picking a Thermal Color Palette* [online]. [Cit. 07.05.2022]. Dostupné z: https://www.flir.com/discover/industrial/picking-a-thermal-color-palette/

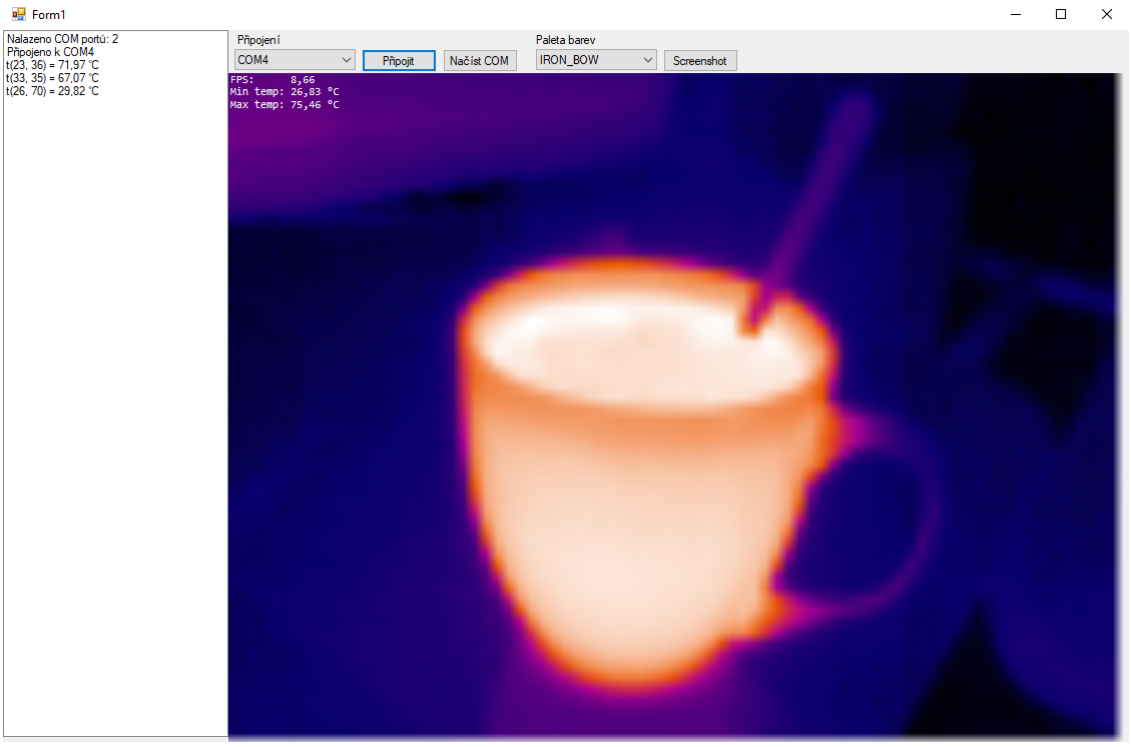
[7] FLIR. *Your Perfect Palette* [online]. [Cit. 07.05.2022]. Dostupné z: https://www.flir.com/discover/ots/outdoor/your-perfect-palette/

Příloha A

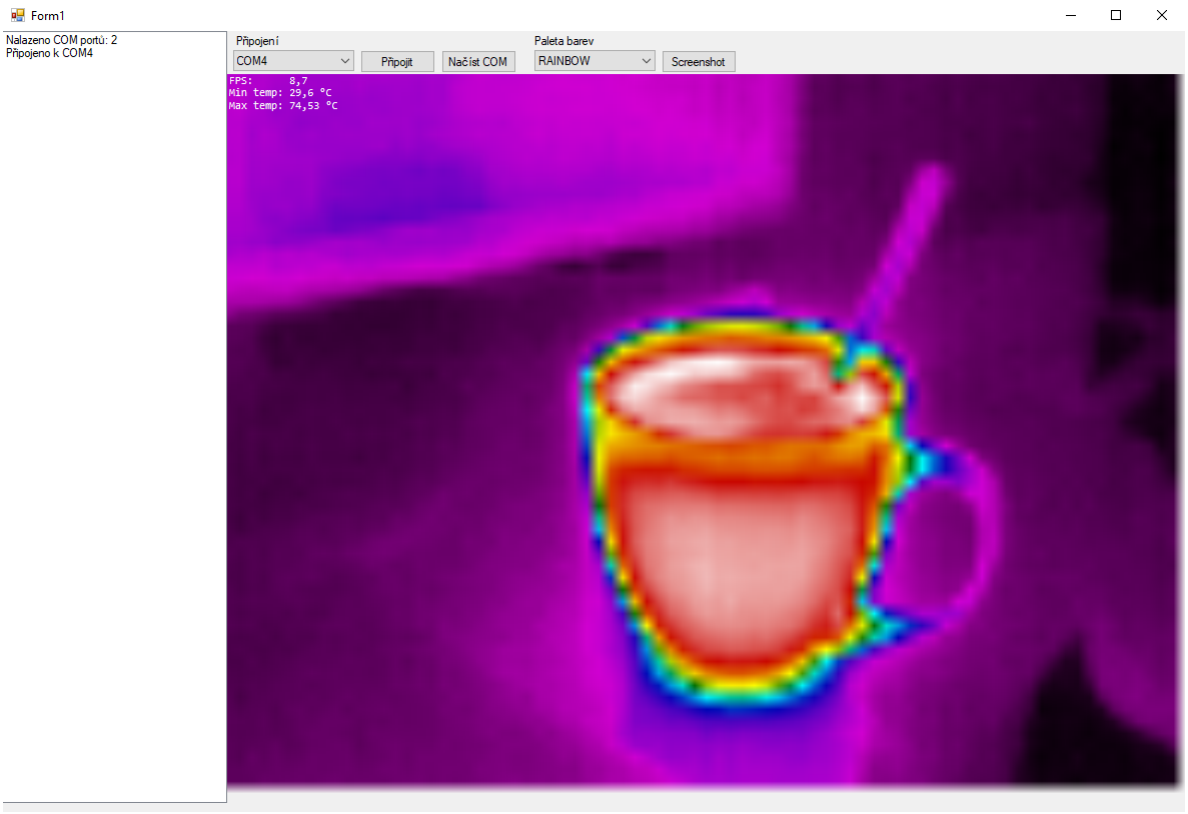
Pořízené snímky



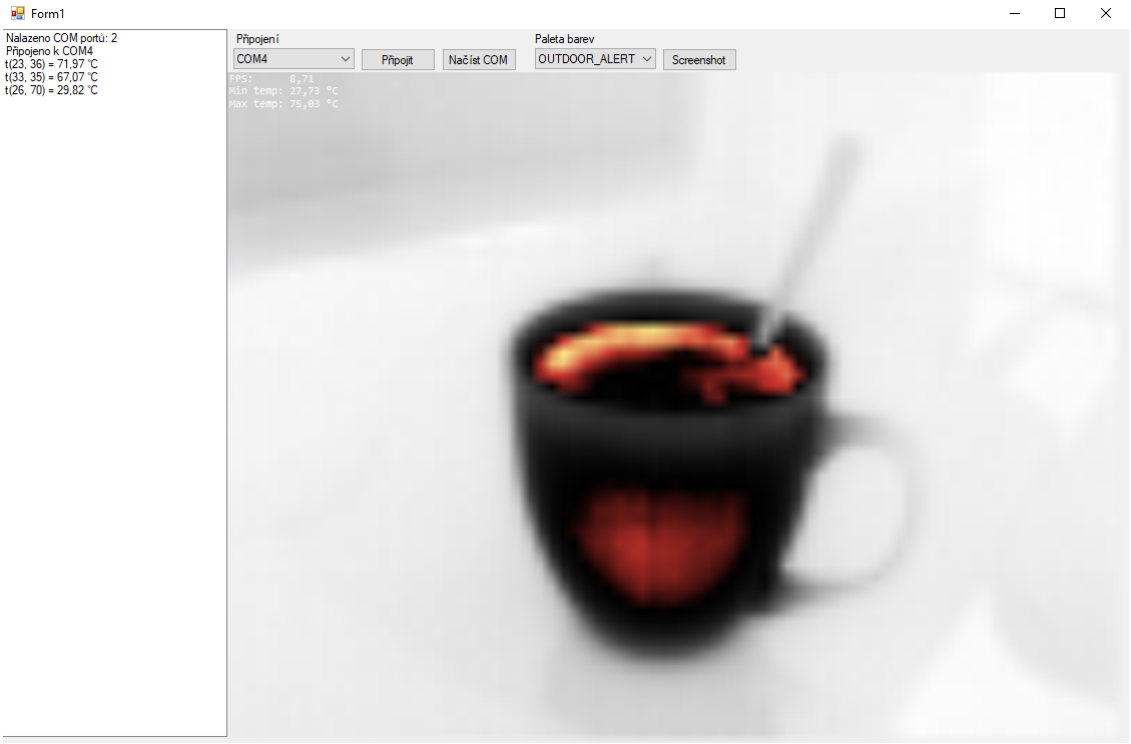
Obr. A.1: Snímek obrazovky aplikace s paletou barev Greyscale



Obr. A.2: Snímek obrazovky aplikace s paletou barev Ironbow



Obr. A.3: Snímek obrazovky aplikace s paletou barev Rainbow



Obr. A.4: Snímek obrazovky aplikace s paletou barev Outdoor alert