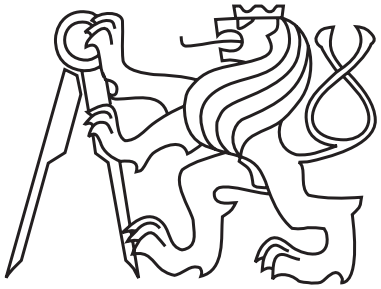
České vysoké učení technické v Praze
Fakulta elektrotechnická



Diplomová práce

# Autonomní stereosnímač pro vyhodnocení polohy překážek

*Bc. Tomáš Kamenický*

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

Studijní program: Kybernetika a robotika

Obor: Senzory a přístrojová technika

květen 2011

ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE
Fakulta elektrotechnická
Katedra měření

Akademický rok **2010-2011**

# ZADÁNÍ DIPLOMOVÉ PRÁCE

Student: **Bc. Tomáš Kamenický**

Program: **Kybernetika a robotika**
Obor: **Senzory a přístrojová technika**

Název tématu česky: **Autonomní stereosnímač pro vyhodnocení polohy překážek**

Název tématu anglicky: **Autonomous Stereosensor for Obstacle Detection**

## Pokyny pro vypracování:

Navrhněte a realizujte autonomní stereosnímač pro vyhodnocení polohy překážek a záznam jejich obrazu. Orientujte se na využití obrazových senzorů CMOS a procesorů řady STM32. Prověřte možnost vícekanálového synchronizovaného záznamu obrazů na paměťové medium za použití protokolu PTP. Navrhněte a implementujte potřebné algoritmy zpracování obrazu. Vytvořte další programy, jako je ovladač pro LabView a řídicí aplikace pro spolupráci senzoru s PC prostřednictvím rozhraní USB.

**Seznam odborné literatury:**

[1] Hlaváč, V., Sedláček, M.: Zpracování signálů a obrazů. Vydavatelství ČVUT, Praha 2009

[2] Fischer, J.: Optoelektronické senzory a videometrie. Skripta ČVUT, Praha 2002

[3] RM0008 - Reference manual, Doc. ID 13902 Rev 11, STMicroelectronics 2010

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

Datum zadání diplomové práce: 5. ledna 2011

Platnost zadání do[1]: 29. června 2012

L.S.

Prof. Ing. Pavel Ripka, CSc.
vedoucí katedry

Prof. Ing. Boris Šimák, CSc.
děkan

V Praze dne 5. ledna 2011

[1] Platnost zadání je omezena na dobu tří následujících semestrů.

## Poděkování

Na tomto místě bych rád poděkoval svým rodičům, přátelům a přítelkyni, kteří mě podporovali po celou dobu realizace této práce.

Dále bych rád poděkoval doc. Ing. Janu Fischerovi, CSc. za vedení diplomové práce, za jeho cenné rady, podklady a připomínky, a především za trpělivost v průběhu řešení.

## Prohlášení

Prohlašuji, že jsem svou diplomovou práci vypracoval samostatně a použil jsem pouze podklady uvedené v přiloženém seznamu.
Nemám závažný důvod proti užití tohoto školního díla ve smyslu §60 Zákona č. 121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon). Tato práce vznikla v laboratoři videometrie, katedry měření ČVUT - FEL v Praze pod vedením doc. Ing. Jana Fischera, CSc. Navazuje též na výzkum v rámci MSM6840770015 - "Výzkum metod a systémů pro měření fyzikálních veličin a zpracování naměřených dat", jehož některé poznatky a výstupy v oblasti optoelektronických senzorů také využívá.

V Praze dne 13.5.2011 .....

## Abstract

This diploma thesis describes the complete design of an autonomous stereo transmitter, which task is to determine the position of obstacles. The system is based on a 32 bit CPU STM32F207 using visual interface DCMI interconnected with DMA to the USB. The diploma thesis describes how the theory of image processing and display as well as hardware and software has done.



## Abstrakt

Tato diplomová práce popisuje kompletní návrh autonomního stereo snímače, kterého úkolem je určení polohy překážek. Systém je založen na 32 bitovém procesoru STM32F207 s využitím obrazového rozhrání, připojení k USB a DMA přenosu. Práce popisuje jak teorii zobrazení a spracování obrazu tak i hardware a software jednotky.

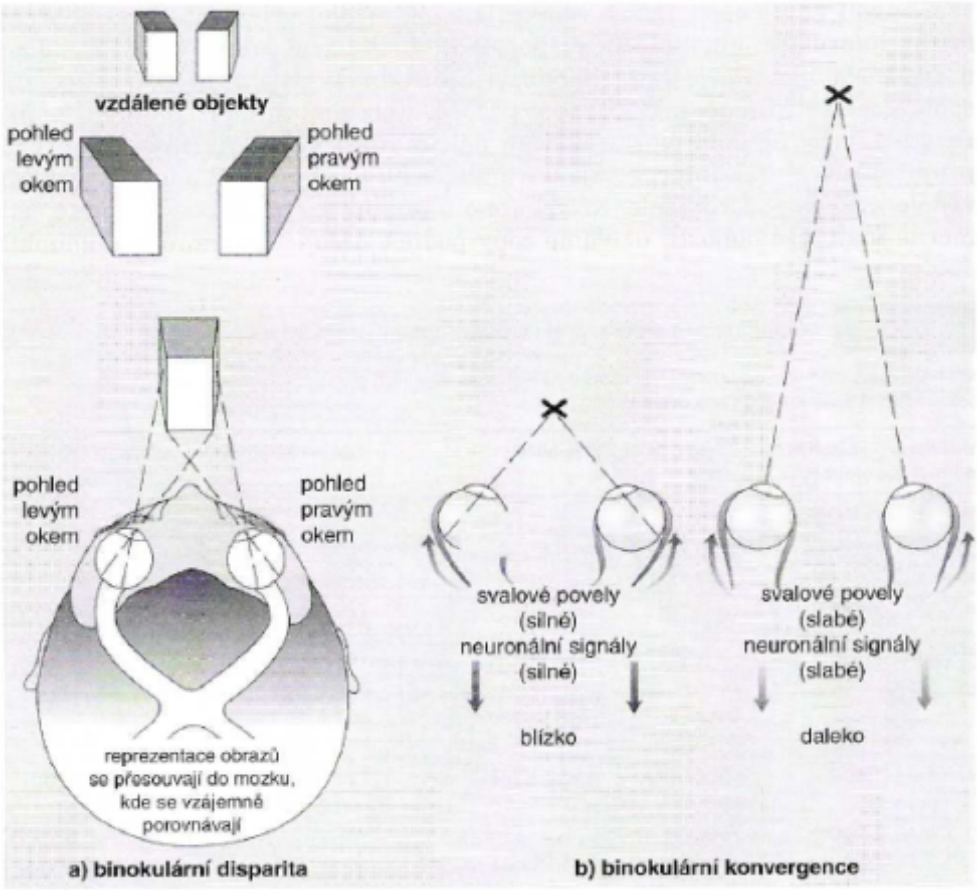
# Obsah

# Seznam obrázků

# Seznam tabulek

# 1 Úvod

Stereo snímač, stereometrie a stereoskopie jsou založeny na prostorovém vjemu, kterému se chci věnovat. Cílem mé práce je sestrojení jednoduchého a levného detektoru překážek na bázi zpracovaní obrazů ze stereo kamer, s pevnou pozicí. Jednotka by měla sloužit jako senzor přiblížení pro robotiku, stereometrii prostoru pro zaměřování a praktické využití pro nevidomé lidi. Inspiraci algoritmů jsem hledal v poznání lidských orgánů, očí a jejich zpracování neuronem čili mozkem. Fyziologické principy prostorového vidění [1, str.50] spočívají v binokulární disparitě a binokulární konvergenci. Binokulární konvergence využívá stáčení očí dovnitř při přibližování předmětu. Úhel stočení je pak úměrný k přiblížení objektu. Díky vzdálenosti mezi očima vidíme objekt z odlišného úhlu. Rozdíl v pohledu dvou očí se nazývá binokulární disparita. Binokulární disparita je užitečná do vzdálenosti tří až čtyř metrů (Atkinson, 2003).

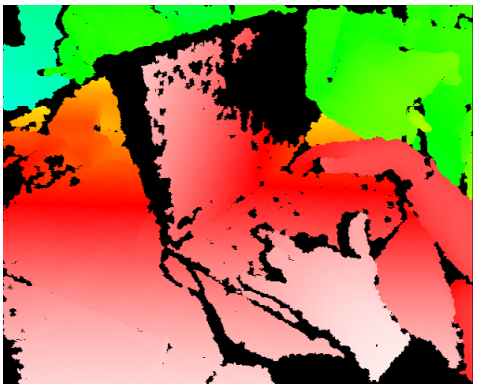



Obr. 1.1: Binokulární vodítka prostorové hloubky (Sternberg, 2002)
a) binokulární disparita b) binokulární konvergence

Z toho důvodu je binokulární disparita vhodná inspirace pro dosažení cíle mojí práce. S rostoucí vzdálenosti mezi kamerami můžeme tak docílit větší citlivost snímače na větší vzdálenost. Dalším cílem mé práce je poukázat na to, že i s relativně levným procesorem bez podpory vektorových počtů lze zpracovat snímky v krátkém okamžiku a podat tak hodnotnou informaci pro nadřazený systém. Problémem zůstává pouze správná implementace řešení a samotné nalezení řešení. Na světě jsou k dispozici různá zařízení pro měření prostoru a poloh. Nejznámější je stereo senzor Kinect od společnosti Microsoft. Tento stereo senzor pro herní konzole Xbox 360 využívá miniaturní infračervený laserový projektor, se kterým osvětluje scénu (obr. 1.2), a na

základě znalosti geometrie laserových paprsků se provádí výpočet pro prostorový vjem (obr. 1.3). Tento údaj pak hry využívají ke svému bezdotykovému ovládání.

Obr. 1.2: Laserové osvětlení Kinect [2]



Obr. 1.3: Hloubková mapa Kinect [2]

Lze tedy vzít levného robota, přidat k němu kameru, software a zprovoznit? Nelze použití klasické metody filtrace a transformace obrazu. Výpočetní výkon je z důvodu jednoduchosti a ceny omezen. Pokud tedy nalezneme nenáročné algoritmy, lze tuto jednoduchou myšlenku úspěšně realizovat.

# 2 Rozvrh práce

V úvodu práce jsem popisoval cíl autonomního stereo snímače, kterého úkolem je detekovat překážku pomocí dvou kamer s využitím 32 bitového procesoru bez podpory vektorových počtů. K dosažení cíle je však nutné zdolat několik úkonů a postupů, které je nutno řešit. Proto jsem se nad zadáním práce zamyslel a vypsal heslovitě seznam klíčových myšlenek, které potřebuji k úspěšnému dosažení cíle.

Diplomová práce je psaná v té samé chronologii jako zmíněný seznam úkonů.

## 2.1 Test algoritmů

- Nalézt vhodné testovací nástroje
- Návrh různých metod zpracování a zhodnocení stávajících metod
- Ověření a simulace algoritmů

## 2.2 Výběr technologie

### 2.2.1 Obrazový senzor

- Nastudovat materiály a dokumentace. Zajištění, jakým způsobem se řídí obrazový senzor.
- Zjistit jaké jsou možnosti přenosu dat a obecné možnosti konfigurace.
- Jaká je nejvyšší možná vzorkovací frekvence

### 2.2.2 Procesor

- Možnost procesoru STM32F107 a STM32F217
- Přenosové a výpočetní rychlosti
- Velikost vnitřní FLASH a RAM paměti
- Možnost využití DMA přenosu
- Interface SDIO pro MMC a SD karty
- Interface USB pro spojení s PC

### 2.2.3 Zapojení

- Jaký typ zapojení se použije (centralizovaný/decentralizovaný).
- Výpočet náročnosti zapojení

### 2.2.4 Programování

- Zvolit vhodné vývojové prostředí
- Možnost debuggingu a nahrávání programu
- Knihovny pro danou platformu

## 2.3 Realizace

### 2.3.1 Návrh hardware

- Návrh procesorové desky s možností osazení řad STM32F10x a STM32F2xx
- Volba napájecího zdroje
- Návrh sběrnic s vysokým kmitočtem pro propojení kamer
- USB rozhrání pro přenos dat
- Jednoduchá linka pro komunikaci
- Oživení jedntoky stereo snímače

### 2.3.2 Software

- Vymyslet uživatelské rozhrání
- Vytvořit náhled obrazů
- Způsob vyčtení parametrů a naměřených dat

### 2.3.3 Firmware

- Správná konfigurace procesoru
- Zavádění periferii a jejich nastavení
- Vymyšlení algoritmů, která nejméně vytěžuje CPU
- Správné využiti IRQ a DMA periferií.

## 2.4 Měření

- Navrhnout postup měření
- Záznam hodnot a jejich zpracování
- Zhodnocení měření podle simulací a předpokladů

## 2.5 Závěr

- Soupis poznatků a výsledků
- Porovnání
- Případné pokračování projektu

# 3 Návrh a volba algoritmů

Při návrhu algoritmů jsem se zaměřil na algoritmy s nízkým nárokem na výpočetní jednotku procesoru. Většina algoritmů pracuje s jedním obrazovým vstupem, ve kterém se snaží hledat překážky. V našem případě však máme obrazové vstupy dva. Drtivá většina patentovaných algoritmů řeší prostorový vjem pomocí různých prostorových transformací. Hledají podobné objekty a podle geometrie kamer propočítávají jejich polohu.

Toto nám ovšem nepomůže, algoritmy jsou výpočetně náročné a složité. Proto jsem se nad problematikou určení polohy zamyslel. Navrhnul jsem celkem dvě metody, které jsem před samotnou implementací do systému otestoval a zhodnotil. K testu algoritmu jsem použil program 3DS Max 9 a Matlab. V Matlabu je naprogramován algoritmus detekce přiblížení a ve 3DS Max-u jsou vytvořené testovací scény s různým pozadím, s různou texturou objektů a konfigurací kamer.

## 3.1 Metoda sumace rozdílových obrazů

Jako první a jednoduchá varianta detekce vzdálenosti je prosté porovnání dvou obrazů pomoci jejích matematického rozdílu. Matematický popis této metody je dán vztahem:

$$sumar = \sum | \begin{bmatrix} \quad obraz1 \quad \end{bmatrix} - \begin{bmatrix} \quad obraz2 \quad \end{bmatrix} |$$

, kde *sumar* je výsledek vztahu, který závisí na globálním prostorovém vjemu. Tuto metodu jsem nazval metodou sumace rozdílových snímků.

Testovaný algoritmus pro Matlab jsem napsal následovně:

```
1.  % Test algoritmu priblizeni ze stereo obrazku
2.  %Uklidime pracovni prostor
3.   clear all;
4.   figure(1);
5.  %celkovy pocet snimku
6.   total_images=100;
7.  %cesta k obrazkum
8.   cesta='obr\\scene 1_koule10cm_120st_100cm\\';

9.   sumar=zeros(total_images);

10. %Projdeme vsechny obrazky
11. for image_number = 1:total_images

12. %Nacteme obrazek pro zpracovani
13. filename=sprintf('%scamera-left%04d.png',cesta,image_number);
14. imleft=imread(filename);
15. filename=sprintf('%scamera-right%04d.png',cesta,image_number);
16. imright=imread(filename);

17. %konverze do odstupne sedi
18. imleft = double(rgb2gray(imleft));
19. imright = double(rgb2gray(imright));

20. %normalizace na rozsah 0-255
21. imleft=uint8((imleft/max(imleft(:)))*255);
22. imright=uint8((imright/max(imright(:)))*255);

23. %Zobrazime nacteny obrazek
```

```
24. if (image_number==floor(total_images/2))
25. subplot(3,3,1),imshow(imleft),title('Levy obrazek');
26. subplot(3,3,2),imshow(imright),title('Pravy obrazek');
27. end;

28. %udelame rozdilovy obraz
29. imdiff=double(abs((int16(imleft))-(int16(imright))));
30. %Normalizujeme 0-255
31. imdiff=uint8(imdiff/max(imdiff(:))*255);

32. %Zobrazime obraz difference
33. if (image_number==floor(total_images/2))
34. subplot(3,3,3),imshow(imdiff),title('Diference obrazku');
35. end

36. %Secteme vsechny jasove hodnoty rozdiloveho obrazu a vysledek
37. %mereni zapiseme do pola sumar
38. sumar(floor(total_images+1-image_number))=sum(imdiff(:))/(320*240);

39. end;

40. %Vykreslime graf namerene vzdalenosti z diference dvou obrazu
41. subplot(2,1,2);
42. plot(sumar);
43. hold off;
```

Jádrem algoritmu je funkce na řádku 38. Čím bude hodnota rozdílu dvou obrazů větší ($imdiff$), tím je celkový součet jasových hodnot také větší ($sumar$). Tato funkce má své maximum a minimum. Přibližováním nebo oddalováním objektu od kamer dojde vlivem transformace obrazu v kamerách s různou polohou k větší respektive k menší hodnotě $sumar$ v algoritmu.

Tento algoritmus se dá provádět při každém nově načteném pixelu bez nutnosti alokace paměti pro celý obraz. Při každém novém pixelu dojde k výpočtu diference a následně k výpočtu součtu. Je ovšem nutné přijímat pixely střídavě. V případě střídání po řádcích, je nutné ukládat obsah alespoň jednoho řádku z jedné z kamer do paměti procesoru. Následně po vyčtení celého snímku se součet podělí celkovým počtem pixelů. Tato hodnota pak vyjadřuje globální prostorový vjem.

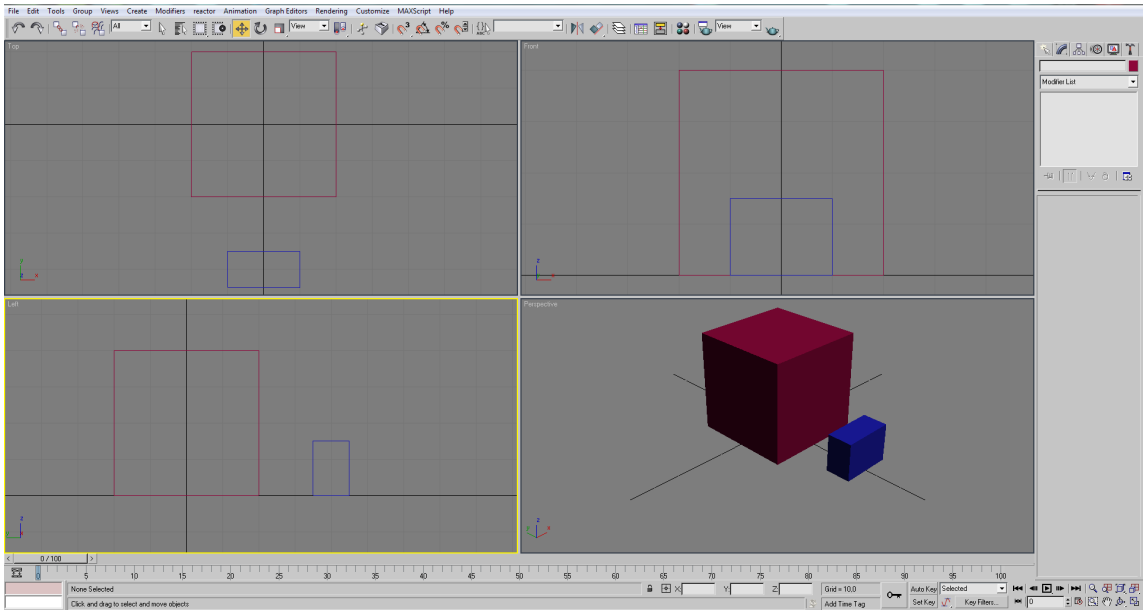
Předpokládané vlastnosti scény, které ovlivňují výsledek algoritmu, jsou: pozadí, velikost, tvar, povrch neboli textura předmětu, osvětlení, úhel pohledu, ostrost obrazu, šířka záběru kamery, souosost a vzdálenost mezi kamerami. Proto si pomocí počítačové grafiky nasimulujeme jednotlivé závislosti zvlášť, abychom si ověřili funkčnost a spolehlivost algoritmu.

## 3.2 3DS Max pro prostorovou grafiku

3DS Max (obr. 3.1) je program pro modelovaní počítačové grafiky a tvorby speciálních efektů ve filmech. Tento program využívá matematické transformace obrazu ze tří rozměrného do dvou rozměrného prostoru. Tento dvou rozměrný prosotor je pak interpretován uživateli, který objekty modeluje přes základní pohledy.

- Top – pohled z vrchu
- Front – pohled zepředu
- Left – pohled z levého boku
- Perspective – Pohled zešikma

V tomto nástroji lze definovat kamery s parametry jako je úhel záběru, hloubka ostrosti (reprezentováno jako speciální efekt) a přesná poloha. Z tohoto důvodu lze namodelovat scénu



Obr. 3.1: Ukázka rozhrání 3DS Max

se dvěma kamerami a nechat spočítat pohled z těchto kamer. Výsledné obrazy zpracujeme v Matlabu. Kombinaci parametrů, které jsem určil, lze nalézt v tabulce 3.1.

| Vzdálenost mezi kamerami [mm] | Úhel záběru[°] | Úhel mezi kamery [°] | Zaostření [mm] |
|---|---|---|---|
| 100 | 120 | 0 | nekonečno |
| 100 | 80 | 0 | nekonečno |
| 100 | 60 | 0 | nekonečno |

Tab. 3.1: Kombinace parametrů pro simulaci

V konfiguracích (tab. 3.1) necháme spočítat obrazy po 100 snímcích, kde každý snímek bude znamenat lineární posun o 1cm ke středu kamer z celkové vzdálenosti 100cm. Jako základní útvar volím kouli zelené barvy s poloměrem 5 cm.

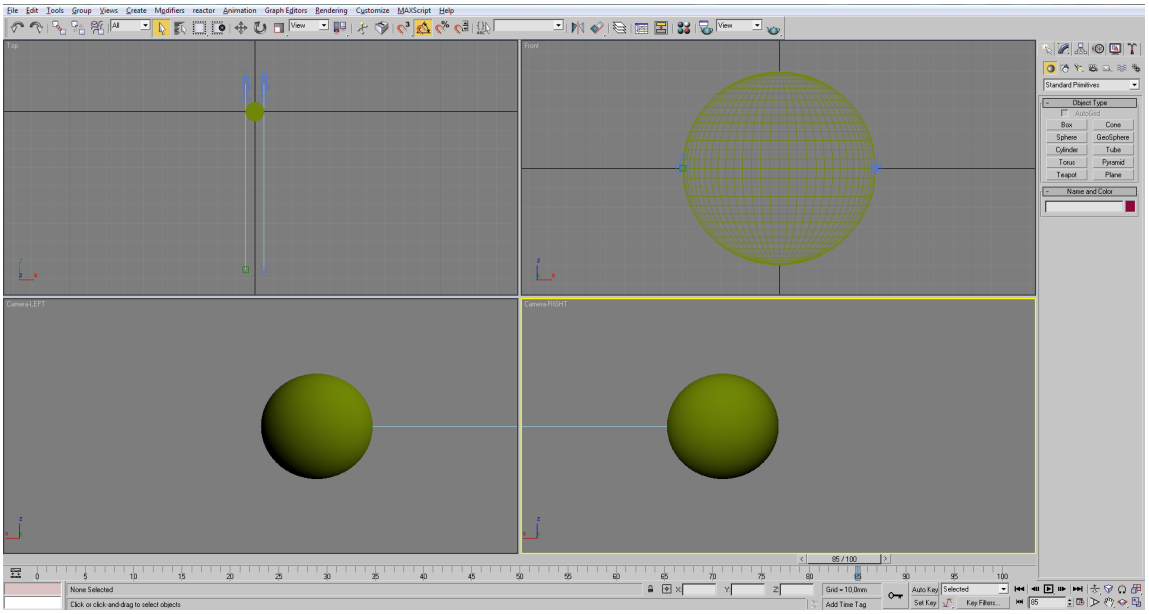
Na obrázku 3.2 vidíme ve spodní části pohled z levé a pravé kamery. Výsledná ukázka vypočtené grafiky je na obrázku 3.3 a 3.4. Z obrázků je patrný i vliv osvětlení scény, tak že klíčovou roli v detekci bude hrát i správné osvětlení.

## 3.3 Test metody sumace rozdílových obrazů

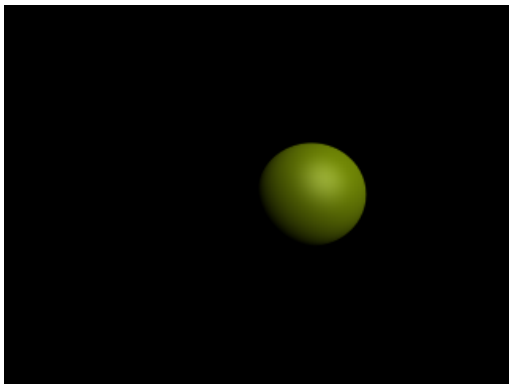
S využitím Matlabu a programu 3DS Max pro prostorovou grafiku jsem nechal jednotlivé konfigurace projít metodou sumace rozdílových obrazů. Výsledky simulace jsem nechal vykreslit do grafů.

### 3.3.1 Scéna 1, porovnání úhlu záběru

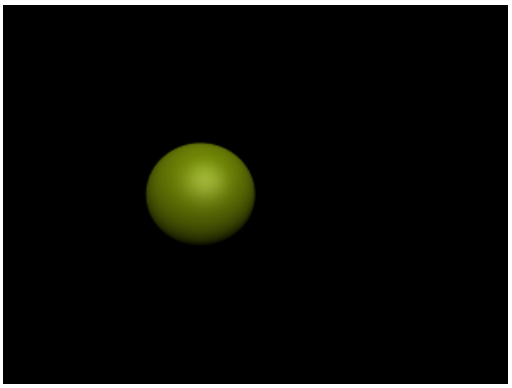
Čím menší je úhel záběru (obr. 3.5), tím kvalitnější a citlivější je detekce vzdálenosti v daném směru. Změnou úhlu záběru, měníme práh detekce vzdálenosti objektu.



Obr. 3.2: Návrh rozložení kamer a objektu v programu 3DS Max



Obr. 3.3: Výsledný výpočet grafiky levého obrázku pro simulaci



Obr. 3.4: Výsledný výpočet grafiky pravého obrázku pro simulaci

### 3.3.2 Scéna 1, porovnání vzdálenosti mezi kamerami

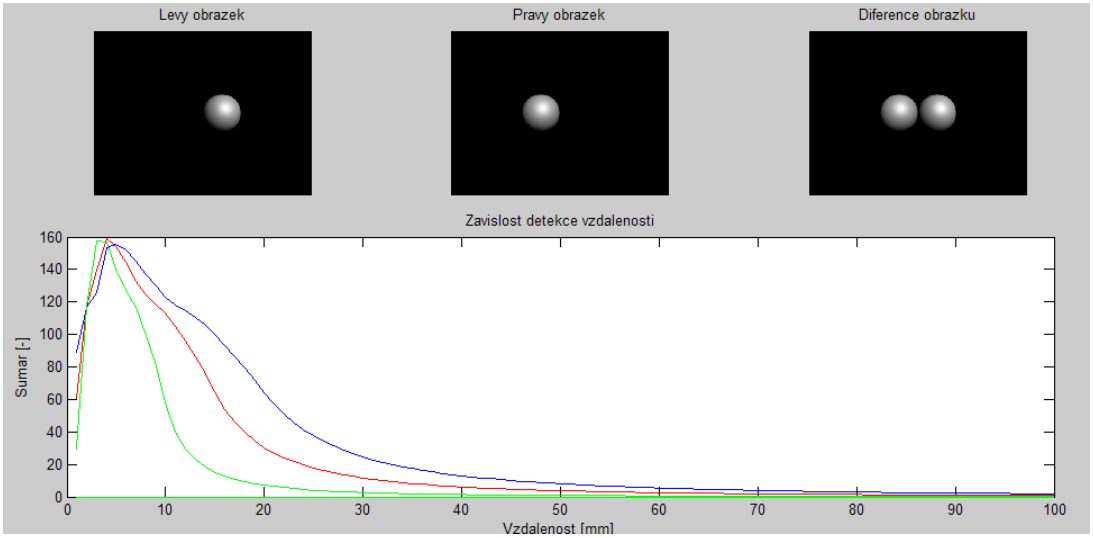
Se změnou vzájemné vzdálenosti kamer dochází ke zvýšení citlivosti na danou velikost objektu (obr. 3.6). Je tedy zřejmé, že pro každou velikost objektu existuje ideální vzdálenost mezi kamerami. V našem případě by vzájemná vzdálenost odpovídala poloměru objektu.

### 3.3.3 Scéna 1, porovnání uhlu mezi kamerami (šilhaní)

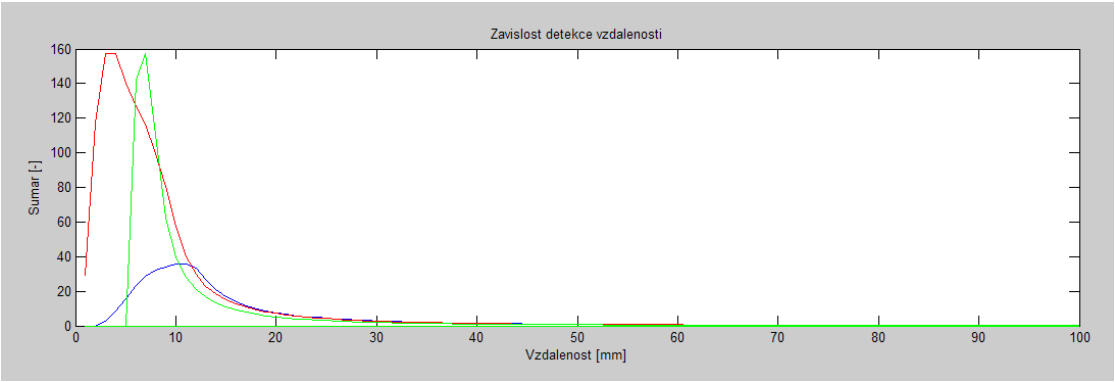
Vzájemný úhel, neboli šilhání kamer, nemá praktický vliv na detekci vzdálenost (obr. 3.7).

### 3.3.4 Scéna 1, porovnání změny ve velikosti objektu

Změna velikosti měřeného objektu má zásadní vliv na detekci vzdálenosti (obr. 3.8). Příliš malé objekty nebudou detekovány vůbec a nadměrně velké objekty budou závislé od jejich povrchu.




Obr. 3.5: Graf, porovnání úhlu záběru
Graf modré barvy odpovídá úhlu záběru 60°. Červené barvy odpovídá úhlu 80° a zelené barvy 120°. Vzdálenost kamer je 10cm.




Obr. 3.6: Graf, porovnání vzdálenosti kamer
Graf modré barvy odpovídá horizontální vzdálenosti mezi kamerami 20cm, červené barvy odpovídá vzdálenosti 10cm a zelené barvy odpovídá vzdálenosti 5cm. Úhel záběru je 120°.

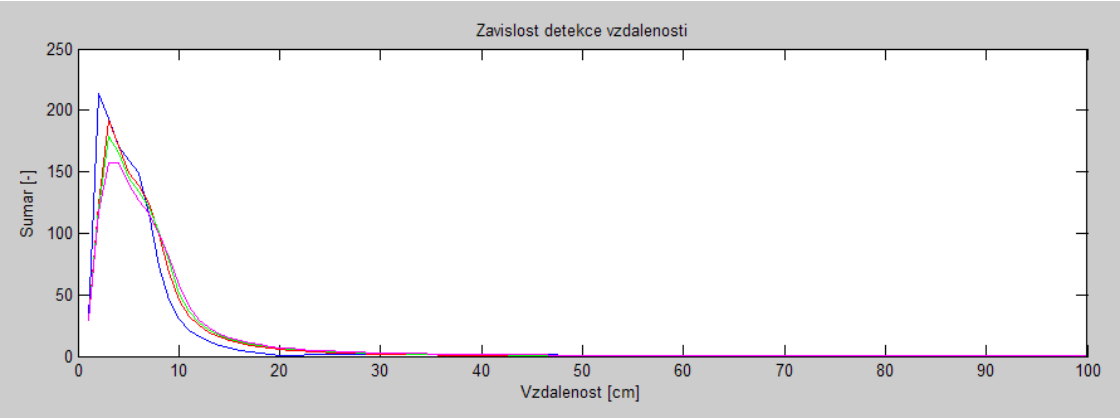
Průběh grafu modré barvy má značný offset vzdálenosti. Tento offset je zapříčiněn poloměrem koule. Odečteme-li tento offset od počátku, pak dojdeme k závěru, že od určité ideální velikosti objektu je měření spolehlivé. Hodnota ideální velikosti bude podle předchozích simulací pouze závislá na vzájemné vzdálenosti kamer a úhlu záběru.

### 3.3.5 Scéna 2 s pozadím

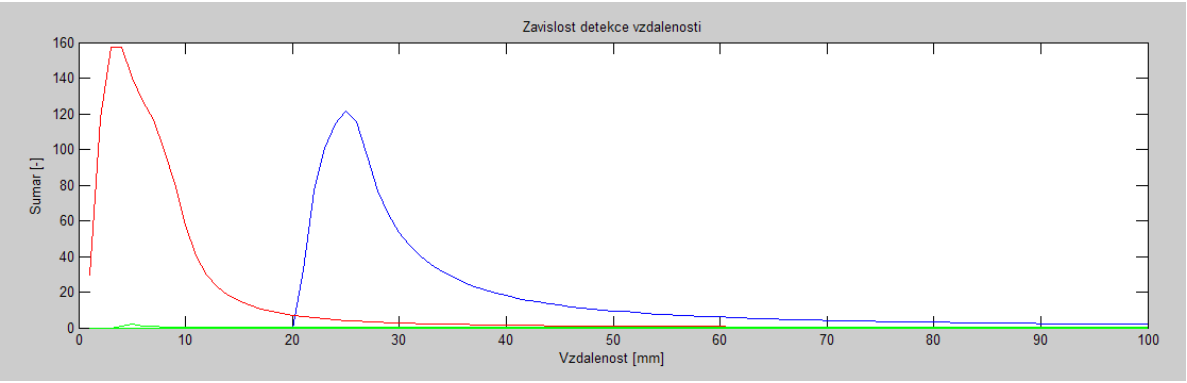
Vliv pozadí je také patrný na průběhu měření a v praxi může znamenat zásadní problém (obr. 3.9).

## 3.4 Metoda lokální disparity obrazů

Základem této metody je zachycení hran v levém a pravém obrazu a následně zjištění disparity (posunu) mezi společnými hranami. Principem zjišťování hran se zabývá několik studií signálů




Obr. 3.7: Graf, porovnání uhlu mezi kamerami
Graf modré barvy odpovídá prúsečníku optických os kamer ve vzdálenosti 20cm, červenou barvou ve vzdálenosti 50cm, zelenou barvou ve vzdálenosti 100cm a purpurové barvy v nekonečnu.




Obr. 3.8: Graf, porovnání změny ve velikosti objektu
Graf modré barvy odpovídá kouli s poloměrem 200mm, červené barvy poloměru 50mm a zelené barvy poloměru 5mm. Úhel záběru kamer je 120° a vzájemná vzdálenost 10cm.

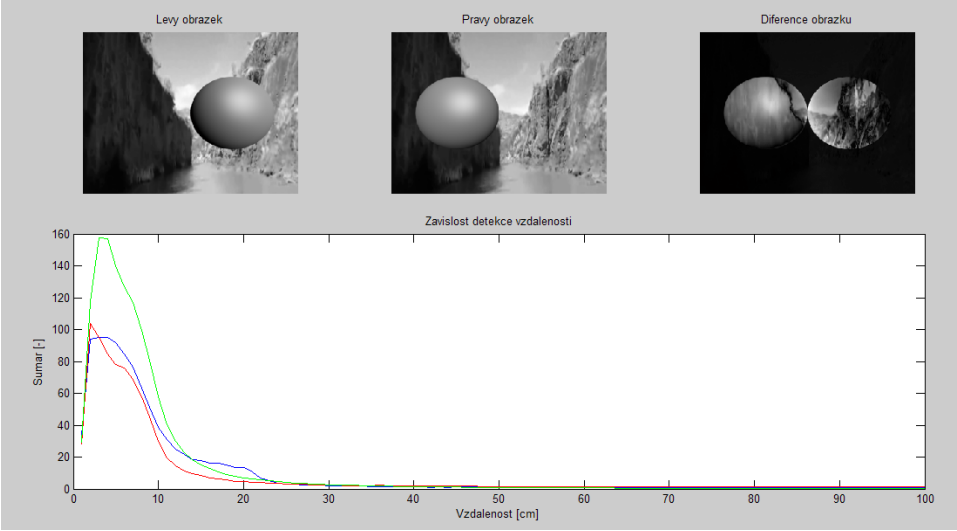
a obrazového zpracování. Nejčastějšími metodami jsou:

- Metoda prahování
- Metoda derivace obrazu
- Metoda filtrace

V našem případě se musíme zaměřit na metody s nejmenší zátěží výpočetní jednotky. Metoda filtrace využívá speciální hodnoty čtvercových matic, které při konvoluci s obrazem vytvářejí obraz derivace původního obrazu. Typickou maticí o rozměru 3x3 pro detekci hran (obr. 3.10) je:

$$T = \begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$$

Nový bod obrazu se vytvoří tak, že z původního bodu a jeho okolí se po vynásobení hodnotami z matic tyto hodnoty sečtou. Tato operace se opakuje pro každý bod obrazu. Je tedy zřejmé, že pro




Obr. 3.9: Graf, porovnání změny pozadí
Graf modré barvy odpovídá pohyblivému pozadí, červené barvy statickému pozadí a zelené barvy pozadí černému.

procesor bez podpory vektorových výpočtů (DSP akcelerátor) bude výpočet takto vzniknutého obrazu trvat mnohonásobně déle. Důsledkem čehož by byla pomalá reakce na případnou detekci překážky.

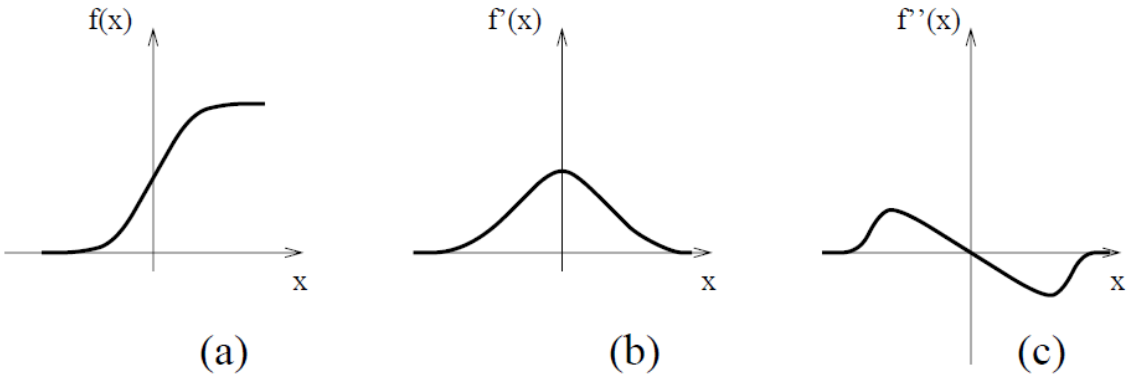
Obr. 3.10: Detekce hran pomocí transformace (zdroj [3])

Rychlejší metodou je metoda derivace obrazu po řádcích. V tomto případě se myslí zpracování jednorozměrného signálu. Diskrétní derivaci lze spočítat jako:

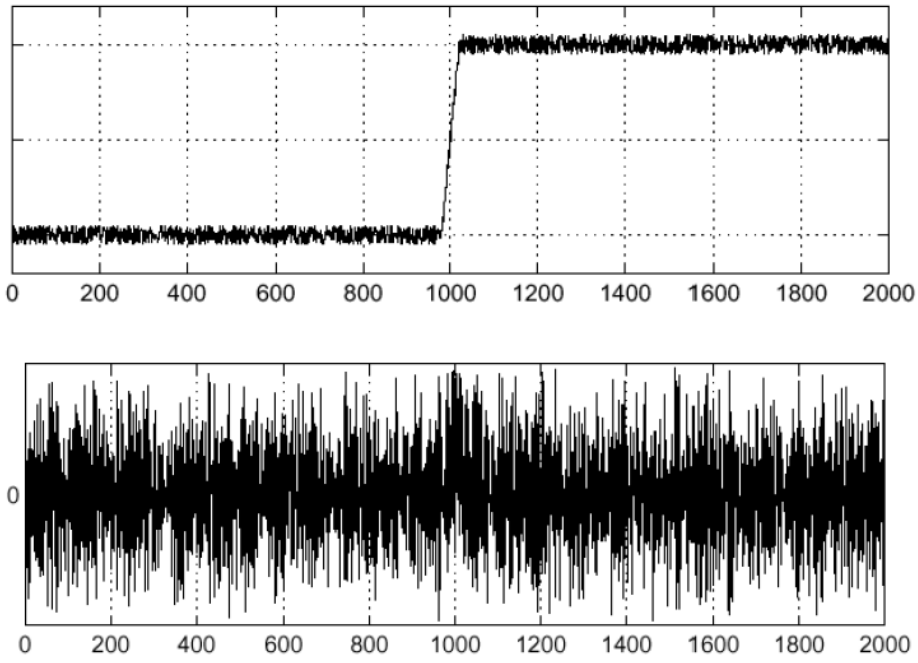
$$NovyBod_{i,j} = Bod_{i,j} - Bod_{i+1,j}$$

Ve své podstatě se jedná také o konvoluci pomocí matice . Aby se ovšem detekovalo snáze, je nutné derivovaný obraz derivovat podruhé (obr. 3.11.c). První derivace obrazu (obr. 3.11.b) totiž hrany zvýrazní a v tomto obrazu nás zajímají lokální maxima jasových hodnot. Pro detekci lokálních maxim použijeme derivaci druhého řádu. Druhá derivace (obr. 3.11.c) nám pak průchodem nulou definuje hledanou hranu. Je ovšem patrné, že pro nalezení hrany touto metodou bychom museli derivovat obraz dvakrát, čili procházet obraz dvakrát.

Další nevýhodou této metody je nutná filtrace signálu (respektive řádku obrazu) před samotnou derivací. V případě výskytu šumu nám derivace nedává jednoznačný výsledek (viz obr.3.12).




Obr. 3.11: Detekce hran pomocí první a druhé derivace (zdroj [4, sld.50])



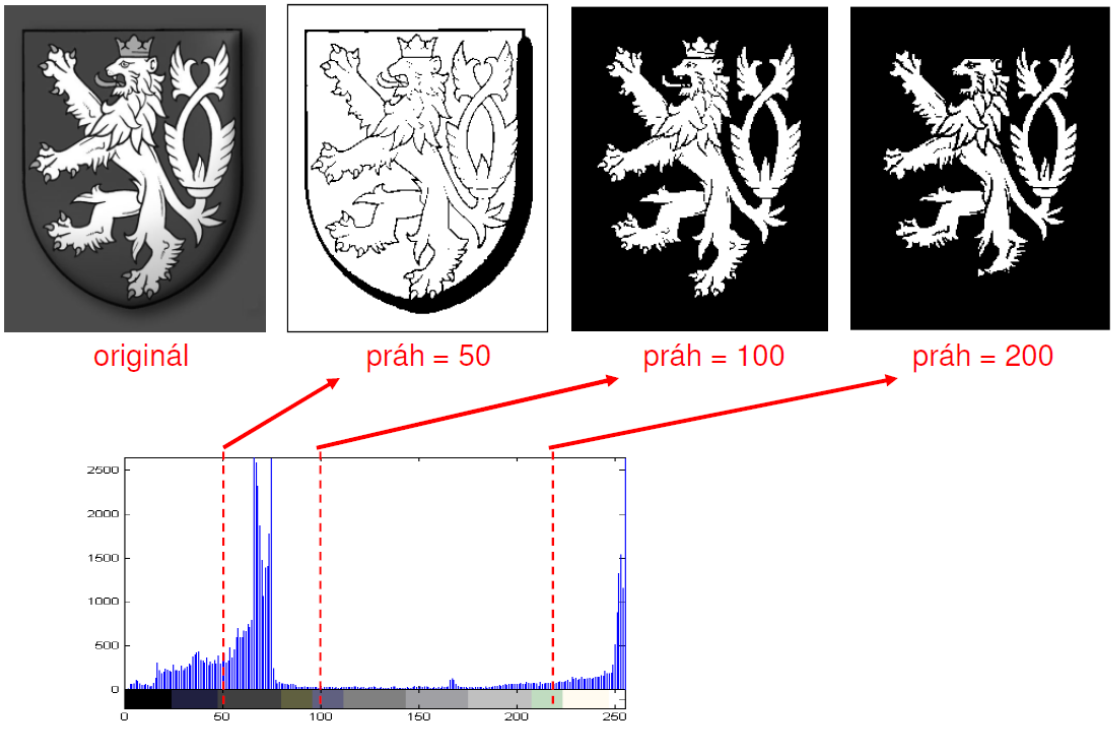

Obr. 3.12: Aplikace derivace na zašumněný signál (zdroj [4, sld.11])

Ještě rychlejší metoda je metoda pomocí prahování. Tato metoda je založena na principu nastavení prahové úrovně tak, aby signál procházející touto prahovou úrovní definoval hranu v obraze. Správné nastavení prahu je kritické viz obr. 3.13.

Většina metod prahování je založená na globální statistice obrazu. V našem případě však máme k dispozici obrazy dva. Levý a pravý. V případě, že vytvoříme nový obraz (obraz diference) odečtením obrazu pravého od obrazu levého, dostaneme dvojí prahování (viz obr. 3.14).

V rozdílovém obrazu jsou patrné dva typy úrovní. Kladná a záporná úroveň jasu. Když si vybereme řez jedním řádkem z obrazu (obr. 3.15), dostaneme průběhy znázornené na obrázku 3.16.

Je nutno si všimnout rozdílu mezi levým a pravým řádkem. Posun průběhu je zapříčiněn geometrickým uspořádáním kamery a hranou objektu. Na řádku diference (obr. 3.16) nám šířka kladného a záporného pulzu určuje vzdálenost překážky. Výhodou této metody je, že v případě vzdáleného pozadí, se jasové hodnoty vzdálených objektů navzájem v obrazu diference vyruší. Stačí tedy pro jednoduchost algoritmu nastavit správné prahování pro kladný a záporný pulz



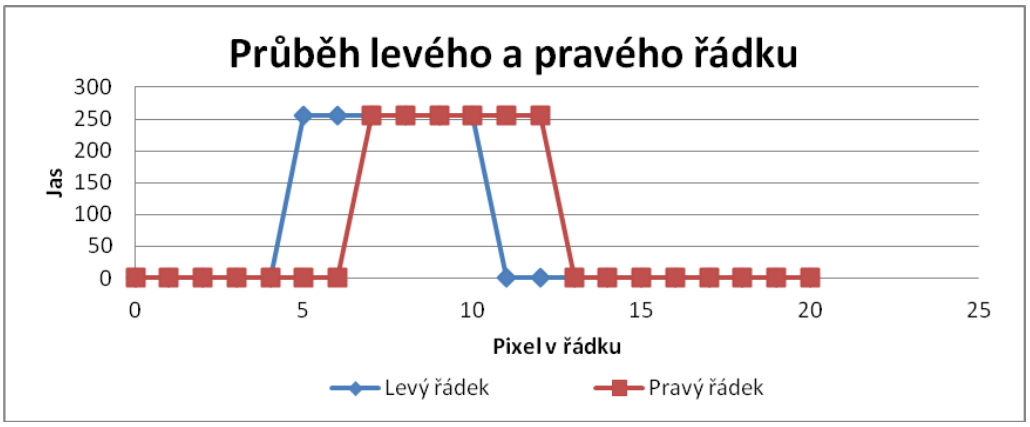

Obr. 3.13: Vliv úrovně prahu při detekci hran (zdroj [5, sld.9])


Obr. 3.14: Obraz diference jednoduché překážky

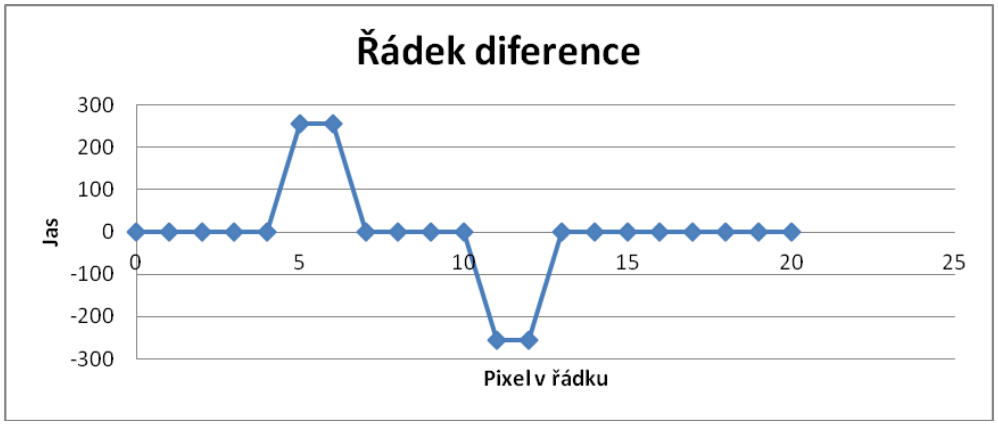
řádku diference. V případě že bude detekovaný objekt příliš blízko, dojde k efektu zdvojeni obrazu (viz obr. 3.17).

V obrazu diference nedojde k překrytí objektů (viz obr. 3.18), a tedy šířka pulzu v diferenciálním řádku neurčuje vzdálenost objektu, ale její skutečnou šířku. Vzdálenost objektu v tomto případě určuje vzdálenost náběžné hrany kladného pulzu a spádové hrany záporného pulzu (viz obr. 3.19).

Je tedy nutné v algoritmu při detekci spádové hrany rozhodnout o kterou ze situací nastává a podle toho upravit výpočet vzdálenosti.

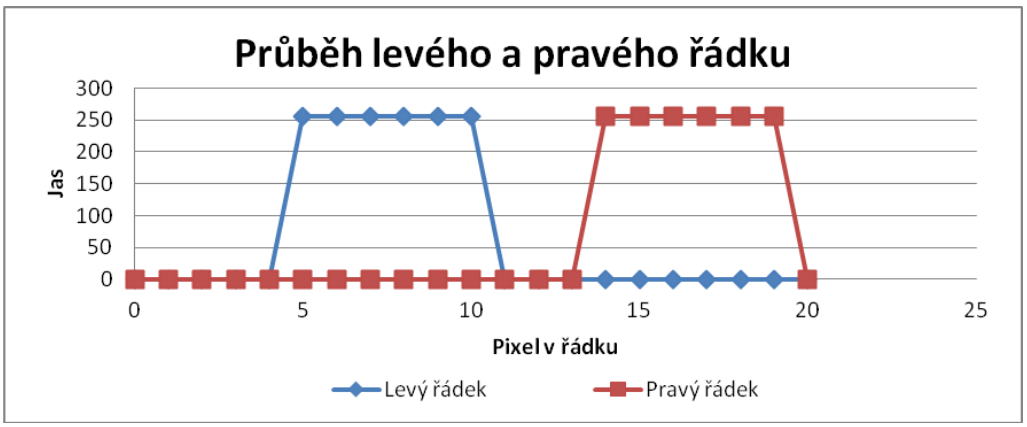



Obr. 3.15: Jasový průbeh v řádku obrazu

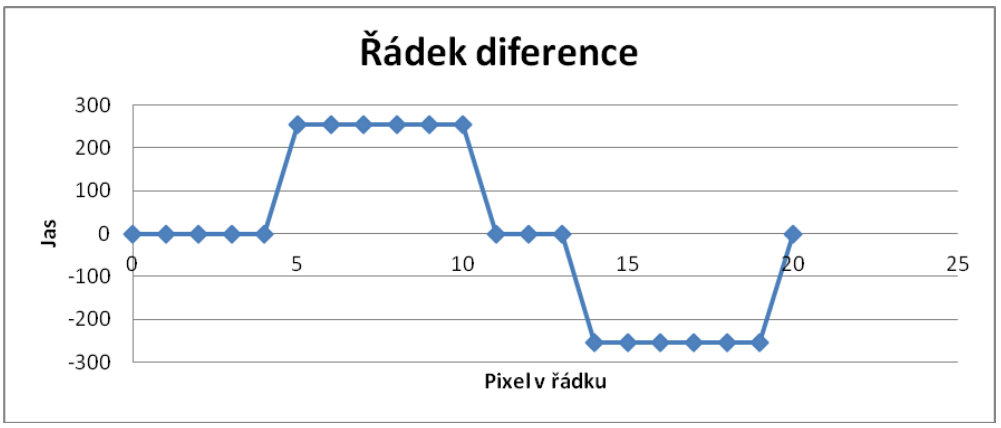



Obr. 3.16: Jasový průbeh v rozdílovém obrazu


Obr. 3.17: Obraz diference jednoduché překážky v malé vzdálenosti




Obr. 3.18: Jasový průbeh v řádku obrazu, překážka v malé vzdálenosti




Obr. 3.19: Jasový průbeh v rozdílovém obrazu, překážka v malé v zdálenosti

# 4 Výběr technologie

Před samotnou realizací projektu je nutné vybrat vhodné součástky a typ zapojení.

## 4.1 Obrazový senzor

### 4.1.1 Teorie

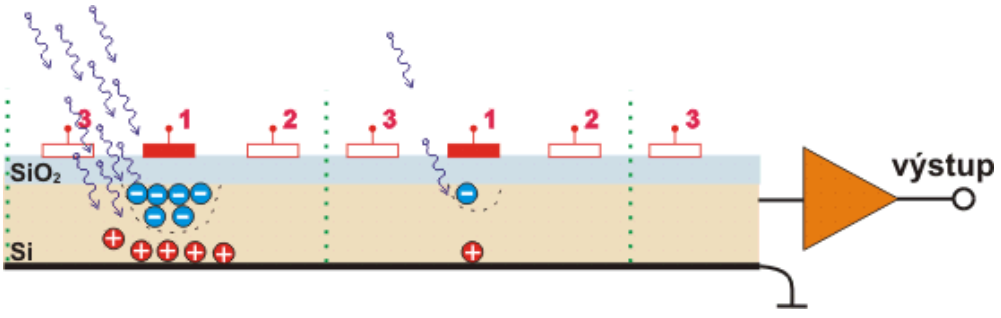
Obrazové senzory převádí světlo na elektrický signál. Podle technologie výroby můžeme senzory rozdělit na CCD a CMOS.

### CCD

Zkratka CCD pochází z anglického Charge-Coupled Device, což v překladu znamená zařízení s vázanými náboji. CCD využívá podobně jako všechny ostatní snímače světlo citlivé součástky fyzikálního jevu známého jako fotoefekt. Tento jev spočívá v tom, že částice světla "foton"při nárazu do atomu dokáže převést některý z jeho elektronů ze základního do tzv. excitovaného stavu. Odevzdá mu přitom energii:

$$E = v.h$$

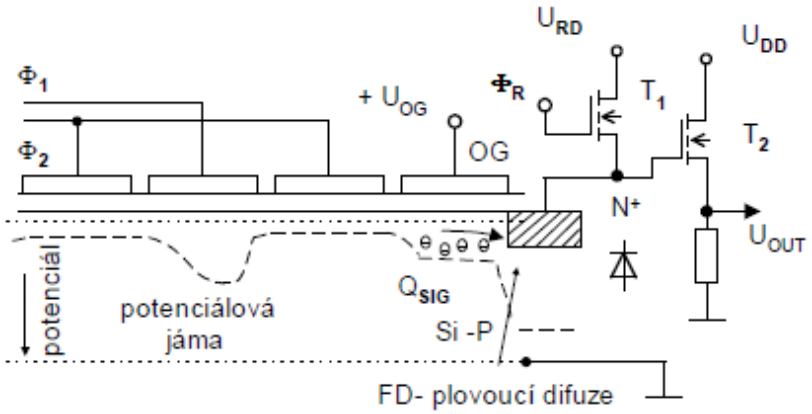
kde $v$ je frekvence fotonu (u viditelného světla v řádu stovek THz) a $h$ je Planckova konstanta. V polovodiči se takto uvolněný elektron může podílet na elektrické vodivosti a je možno ho z polovodiče odvést pomocí přiložených elektrod, tak, jak je to například u běžné fotodiody. Ta proto po dopadu světla generuje elektrický proud. Stejně fungují i fotočlánky, které se používají jako zdroj elektrické energie. U CCD je ovšem elektroda od polovodiče izolována tenoučkou vrstvičkou oxidu křemičitého SiO2, který se chová jako dokonalý elektrický izolant, takže fotoefektem uvolněné elektrony nemohou být odvedeny pryč.



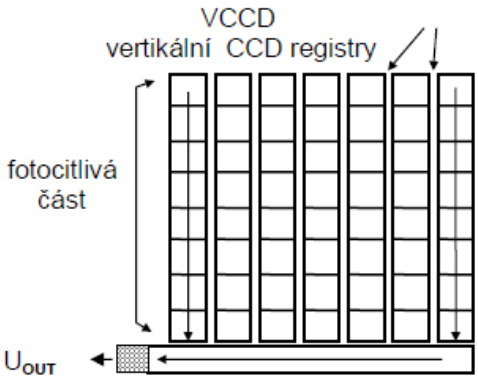

Obr. 4.1: Princip CCD snímače (zdroj [6])

Nahromaděná energie se odvádí pomocí třífázového signálu na elektrodách 1, 2 a 3 (obr. 4.1). Na převod energie se používá převodník náboje Q na napětí U (obr. 4.2). CCD snímač je ve své podstatě analogový posuvný registr.

Jednotlivé pixely tvoří matici (obr. 4.3). Náboje jsou po sloupcích přelévány do posledního řádku. Na tomto řádku je pak umístěn převodník Q na U.



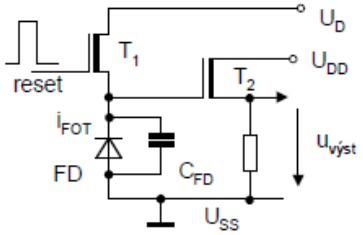

Obr. 4.2: Převodník náboje na napětí (zdroj [7])




Obr. 4.3: Maticové zapojení CCD snímače (zdroj [7])

## CMOS

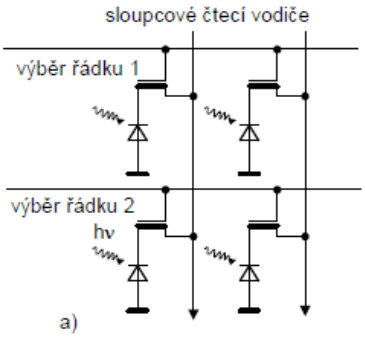
CMOS senzor využívá na rozdíl od CCD snímače fotodiodový proud k vybíjení přednabitého kondenzátoru $C_{FD}$ na obr. 4.4.




Obr. 4.4: CMOS senzor, snímací fotodioda (zdroj [7])

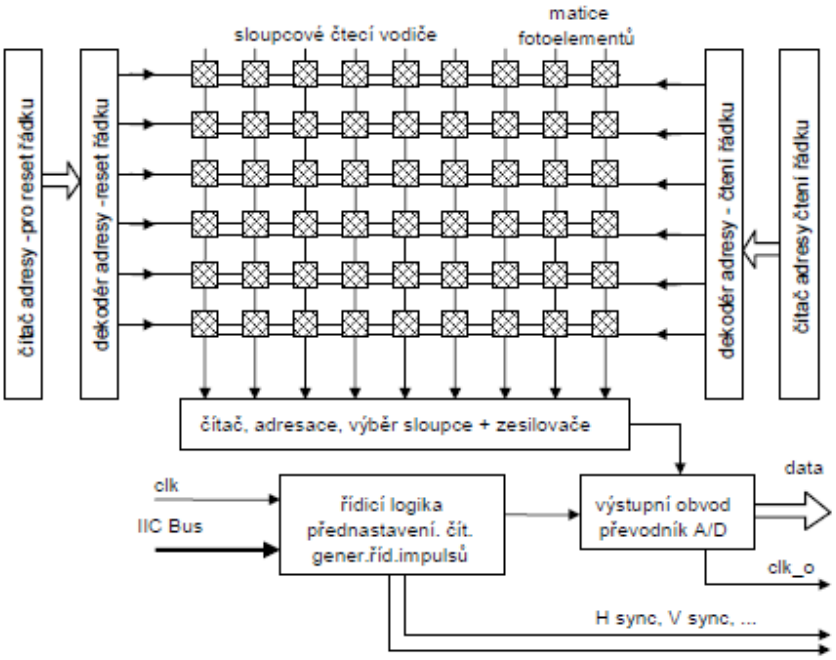
Proud generovaný fotodiodou $FD$ je závislý na intenzitě osvětlení. Zůstatkový náboj na kapacitě vytváří napětí, které otvírá unipolární tranzistor $T_2$. Ten pomocí odporu zesiluje toto napětí pro další zpracování. Jednotlivé pixely jsou zapojeny obdobně jako buňky v paměti RAM (obr. 4.5).

Řídící logika snímače adresuje příslušný řádek a tím aktivuje výstup pixelů (obr. 4.6). Výstupní napětí z pixelů je pak podle nastavení snímače zesíleno a převedeno A/D převodníkem na




Obr. 4.5: CMOS senzor, zapojení bunek (zdroj [7])

digitální výstup.



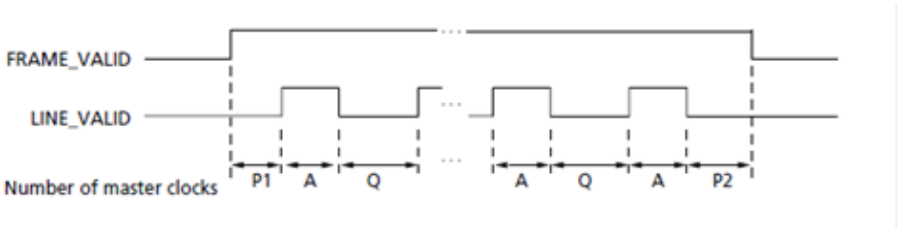

Obr. 4.6: CMOS senzor, vnitřní blokové zapojení (zdroj [7])

### 4.1.2 MT9M001

MT9M001 je CMOS senzor s rozlišením 1280x1024 pixelů. Při maximálním vzorkovacím kmitočtu 48MHz vygeneruje 30 snímků za sekundu. V první fázi senzor načte černé pixely a provede automatickou kalibraci. Poté se synchronně s výstupními hodinami objeví na výstupu data. Krom dat se objeví také informace o validním řádku a validním snímku (obr. 4.7). Pro nás je důležitý stav kdy $LINE\_VALID$ a $FRAME\_VALID$ jsou ve vysoké úrovni.

Senzor se řídí I2C sběrnicí. Řídící logika je navržena tak, že má dvě adresy. Jendu adresu 0xBB pro čtení (sekvence čtení) a jednu adresu 0xBA pro zápis (sekvence zápisu). Přesněji řečeno o čtení nebo o zápisu rozhoduje bit v adrese s nejmenší váhou.

Rozlišení obrazu lze měnit automatickým přeskočením sloupce nebo řádku. Nastavením registru 0x20 na hodnotu 3 povolíme tzv. „COLUMN skip“ a na výstupu dostaneme pouze ty sloupce,

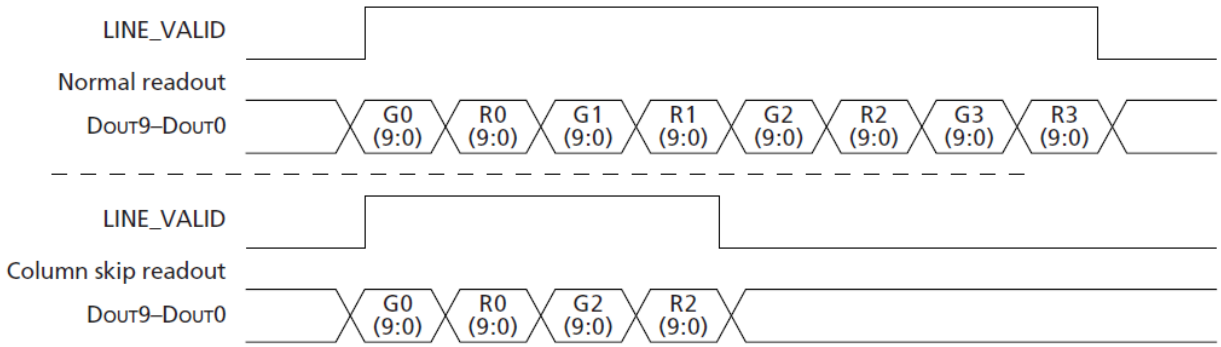



Obr. 4.7: Průběhy řídicích signálů při vyčítání obrazu z CMOS senzoru

které mají bit na pozici 1 rovno 0. Příklad takového nastavení je v tabulce 4.1.

| Sloupec č. | V binární formě | Výčet |
|---|---|---|
| 0 | 0000 0000 | ANO |
| 1 | 0000 0001 | ANO |
| 2 | 0000 0010 | NE |
| 3 | 0000 0011 | NE |
| 4 | 0000 0100 | ANO |
| 5 | 0000 0101 | ANO |
| ... | ... | ... |
| 128 | 1000 0000 | ANO |

Tab. 4.1: Ukázka vynechání sloupců senzoru CMOS




Obr. 4.8: Průběhy řídicích signálů při přeskakování sloupců z CMOS senzoru

G2 a R2 na obrázku 4.8 jsou sloupce číslo 4 a 5 v tabulce 4.1. Znaky G a R značí barvu citlivosti RED a GREEN čili červená a zelená.

Nastavením registru 0x20 na hodnotu 4 povolíme tzv.“ROW skip“ a na výstupu dostaneme, obdobně jako pro sloupce, řádky 0, 1, 4, 5...atd. Celkový počet řádků a sloupců k vyčtení je nastaven v registru 0x03 a 0x04. Při aktivní funkci „ROW a COLUMN skip“ se vyčte pouze polovina řádků definována v registrech 0x03 a 0x04. K nastavení rozlišení se dá ještě použít registr 0x1E. Tento registr slouží pro okénkování (určení aktivní oblasti snímání).

### 4.1.3 MT9V032

MT9V032 je CMOS senzor s rozlišením 752x480 pixelů. Při maximálním vzorkovacím kmitočtem 26,6MHz vygeneruje 60 snímků za sekundu. Časový průběh výstupu dat je obdobný se senzorem

MT9M001. Díky jiné maximální hodinové frekvence se mění i doby integrace a vzorkovaní. Senzor má kromě výstupu paralelních dat k dispozici i výstup LVDS (sériová dvoulinka), která při dodržení správné impedance vedení může dosahovat až délky 8m. Díky tomuto rozhrání lze kombinovat dva senzory (stereoskopická topologie) a propojit je po jednom vedení. V takovém případe se délka LVDS linky zkrátí na 5m. Senzor lze ovládat I2C linkou obdobně jako předchozí senzor MT9M001. Rozlišení obrazu (ROW a COLUMN binning) lze měnit také obdobně na registrech 0x0D, 0x03 a x0x4.

### 4.1.4 Výběr obrazového senzoru

Zvolil jsem CMOS senzor MT9V032. Důvodem je nižší rozlišení (menší náročnost na paměť v případě ukládání snímků) a sériové LVDS rozhraní. Navíc jsem mněl k dispozici hotový modul s obrazovým rozhraním.

## 4.2 Procesor

### 4.2.1 Programové řízení vývodů procesoru STM32F107

Abychom zajistili správnou funkčnost CMOS senzoru, je potřeba dodržet minimální frekvence hodin. Obvykle je tato hodnota větší než 10MHz a z toho plyne i tok dat z obrazového snímače. Procesor řady STM32f10x nemá obrazový interface. Řídící a vstupní signály jsou obsluhovány softwarově. Abychom ale docílili dostatečné rychlosti komunikace, je potřeba procesor podrobit testu, při kterém si změříme maximální rychlosti ovládání vstupů a výstupů. Hodiny pro obrazový senzor lze generovat pomocí časovačů v procesoru. Paralelní přenos dat lze připojit k jednomu vstupu GPIO (PA, PB, PC...atd.) a tím lze softwarově vyčíst až 16 bitů na jednu instrukci. Je nezbytné, aby vyčítaní dat bylo synchronizované s hodinami (pixel clock) ze senzoru.

#### Generování výstupního hodinového signálu

Následující program generuje hodinový signál na pinu 6 portu C:

```
while(1)
{
    GPIOC->BSRR = GPIO_Pin_6;
    GPIOC->BRR = GPIO_Pin_6;
}
```

Port C je nastaven v režimu „PUSH-PULL“ v módu „50MHz“. Kmitočet sběrnice periferie a procesoru je 72MHz.

Po připojení sondy osciloskopu na pin 6 portu C byly naměřeny následující hodnoty: Se zapnutou optimalizací byl kmitočet signálu 12MHz. Bez zapnuté optimalizaci kódu byl kmitočet 4MHz, kde doba trvání logické jedničky byla 27ns. Z toho plyne nutnost psaní kritických částí kódu v asembleru.

#### Čtení dat z portu

Následující program čte vstup pinu 6 portu C:

```
uint32_t ones = 0;
uint32_t zeros = 0;
while (!(GPIOC->IDR & GPIO_Pin_6));

while (GPIOC->IDR & GPIO_Pin_6) {
                ones++;
        }
while (!(GPIOC->IDR & GPIO_Pin_6)) {
                zeros++;
        }
while (1) {};
```

Port C je nastaven jako „floating input“ v módu „50MHz“. Kmitočet sběrnice periferie a procesoru je 72MHz.

Na pin 6 portu C byl přiveden signál o hodnotě 3,3 V a kmitočtu 1kHz se střídou 0,5. Po spuštění programu bylo odečteno 1801 vzorků jedniček (*ones*) a 1892 vzorků nul (*zeros*) bez optimalizace kódu.

Vzorkovací kmitočet spočítame jako:

$$f_{vz} = \frac{celkovy\_pocet\_vzorku}{doba\_mereni} = \frac{1801 + 1892}{\frac{1}{1kHz}} = \frac{3693}{1ms} = 3.7MHz \tag{4.1}$$

Optimalizace kódu v tomto případě selhala. Kompilátor po překladu vyřadil funkci pro počítaní vzorků. Proto se nelze spoléhat na zapnutou optimalizaci při překladu. Řešením není ani proměnná typu *volatile*, neboť jí pak kompilátor neoptimalizuje.

S vhodnou programovací technikou (inline asembler) jsme schopni přenášet data ze senzoru na teoretickém kmitočtu 12MHz. V případě že se nestíhají přenášené data zapisovat, můžem senzor nastavit tak, aby každý další přenášený byte se nastavil až na n-tou periodu hodinového signálu.

### 4.2.2 Procesor STM32F207/217

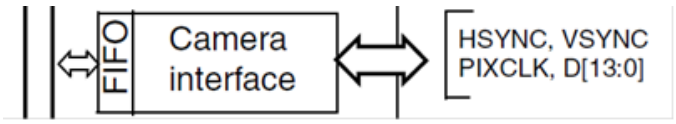
V čase psaní této diplomové práce byl procesor STM32F2xx nově uveden na trh. Je založen obdobně jako předcházející řada na jádře ARM Cortex-M3. Obsahuje až 1Mbyte paměti flash a 128kB paměti RAM. Procesor je taktován na frekvenci 120MHz. Má periferie obsažené v řadě 103 a 107. Nově obsahuje i True random number generator a dále rozhraní High speed USB a Digital camera interface.

### 4.2.3 Kamerové rozhraní procesoru STM32F207

Procesor řady STM32f2 je nástupce předchozí řady 100, vychází také z jeho konceptu a má mnoho společných prvků a dalších vylepšení. Jednou z nich je i obrazové rozhrání DCMI (obr. 4.9), díky kterému lze přímo připojit senzor CMOS k procesoru bez nutnosti složité softwarové obsluhy. DCMI rozhraní obsahuje signály HSYNC, VSYNC, PIXCLK a 14bitů datové sběrnice. Podporuje vnitřní přenos dat pomocí přímého přístupu do paměti (DMA). Vnitřní architektura nové řady procesoru je k dispozici jako elektronická příloha této diplomové práce.

Ke správné funkci senzoru MT9V032 ve spolupráci s procesorem STM32F207 je navíc nutné generovat hodinový signál o frekvenci v rozsahu 13 až 27MHz. K tomu účelu postačí čítač timer1 nastavený jako dělič vnitřních hodin na frekvenci 120MHz.

Tento procesor je diky pro větší paměť a obrazový interface vhodný k našim účelům.

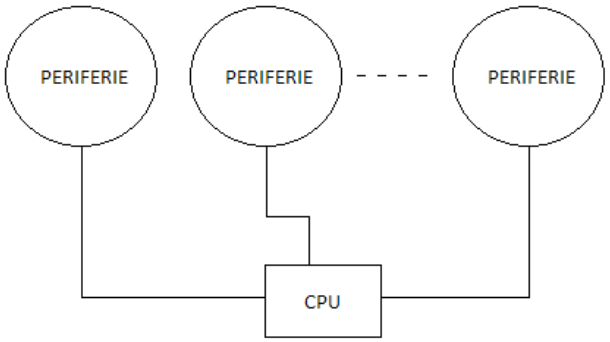



Obr. 4.9: Rozhrání DCMI procesoru STM32F207

## 4.3 Zapojení

### 4.3.1 Typ zapojení a její propustnost

Zapojení rozdělujeme na tzv. centralizované a decentralizované. Na obrázku 4.10 vidíme typické centralizované zapojení, kde periferii představuje obrazový senzor a CPU je jednotka pro zpracováni dat.




Obr. 4.10: Centralizované zapojení

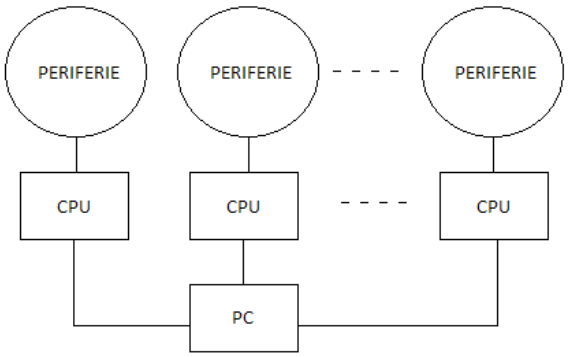
V našem případě je nevýhodou tohoto zapojení náročnost na sběrnici mezi periferií a procesorem. Datovou propustnost lze spočítat jako:

$$\begin{aligned} BW &= pocet\_bodu.barevna\_hloubka.fps \\ BW &= 320.240.8.10 \\ BW &= 76800.8.10\frac{bitu}{sekundu} \\ BW &= 6Mb/s \end{aligned} \tag{4.2}$$

Pro nekomprimovaný obraz o rozlišení 320x240 v 8 bitové stupni šedi a se snímkovací frekvencí 10 obrázků za vteřinu dostaneme datovou propustnost 6Mb/s (v této propustnosti ale není započtena režie a fakt, že CMOS senzor nejdříve obraz snímá a pak teprve přenáší, což navyšuje rychlost přenášených dat z důvodu rezerv pro dobu integrace (snímaní obrazu)). Sběrnice CMOS senzoru se skládá z 10 bitů data, 2 bitů I2C, 4 bitů režie (HSYNC, VSYNC, PCLK, SCLK). Ze senzoru využijeme pouze 8 bitů dat (nižší 2 bity zahodíme). Centralizovaný režim nám nevyhovuje i z toho důvodu, že každá periferie může být od jiného výrobce a tedy může taky vyžadovat jiné řízení. To značně komplikuje nároky na software a systém jako takový.

Výhodnější je pro nás tedy zapojení decentralizované (obr. 4.11).

Z pohledu celého systému nám zvyšuje modulárnost a zjednodušuje implementaci. Zátěž je rozdělena na každou jednotku zvlášť.




Obr. 4.11: Decentralizované zapojení

Na místo každé periferie zapojíme dvě kamery spojené se společnou datovou sběrnicí. Tuto sběrnici pak připojíme do obrazového rozhrání procesoru.

## 4.4 Volba vývojového programového prostředí - IDE

Pro procesory s jádrem CORTEX-M3 existuje mnoho vývojových prostředků jako RAISONANCE, ATTOLIC True studio, KEIL anebo HITEX. Všechny tyto vývojové prostředí mají společné následující body:

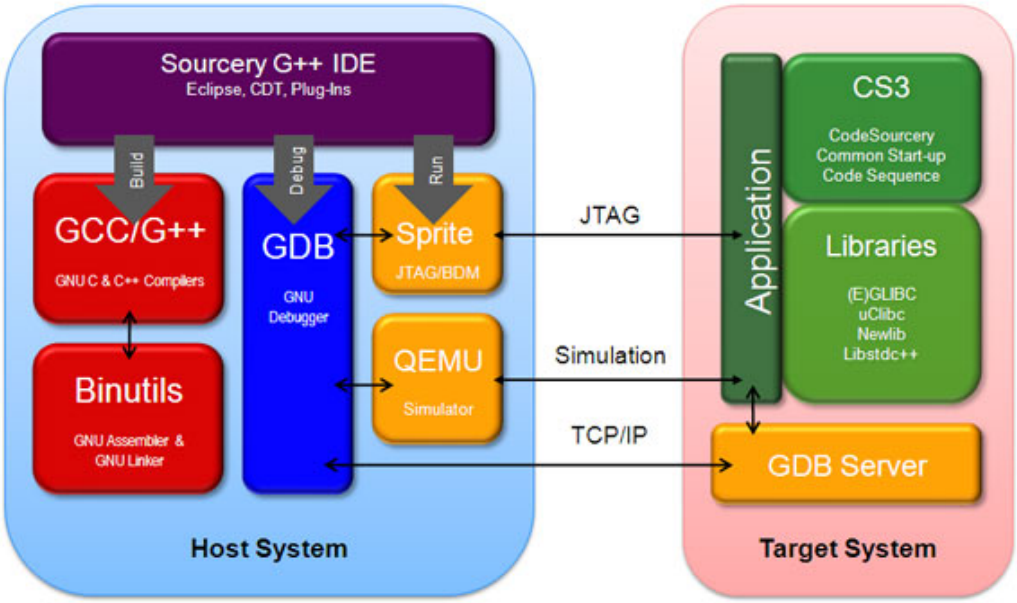
- Inteligentní textový editor s opravou syntaxe
- Překladač (C nebo assembler)
- Linker
- Debugger

Všechny tyto nástroje jsou ale od výrobců třetích stran a tomu odpovídá jejích cena. Proto jako alternativa těchto nástrojů poslouží textový editor Eclipse včetně podpory překladače GCC od společnosti Codesourcery. Překladač funguje bez omezení velikosti kódu a má také možnosti optimalizace kódu.

Ladění programu se pak provádí pomocí GDB serveru (obr. 4.12), který je podporován textovým editorem Eclipse. Pomocí programátoru ST-LINK se fyzicky propojí vyvíjená jednotka s vývojovým prostředím. K fyzickému propojení slouží rozhraní JTAG nebo SWD na procesoru.

Další nedílnou součástí programování jsou knihovny, reprezentující standardní funkce. Pro vestavěné aplikace je vhodná knihovna *newlib* která podporuje širokou škálu architektur procesorů se standardními funkcemi jako $printf$, $flush$, $get$ nebo $fopen$. K periferiím procesoru slouží knihovny dodané od výrobce. Ke knihovnám periferií jsou napsány i příklady použití, které lze aplikovat v praxi.

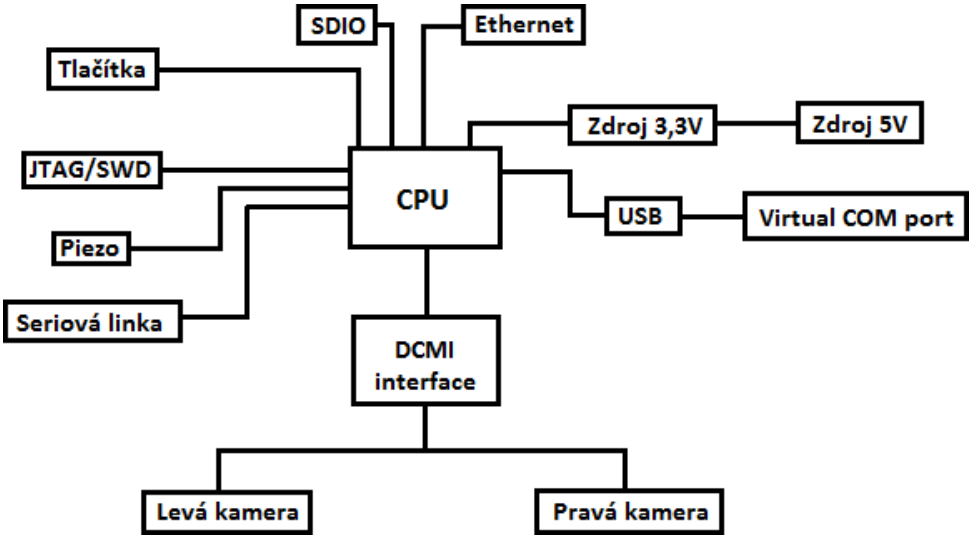
Původně se mělo použít vývojové rozhrání Eclipse, které však v době psaní DP nepodporovalo procesory řady STM32F2xx. Ty byly podporovány pouze ve vývojové prostředí KEIL. Je nutné stáhnout nejnovější ovladač k programátoru a přepsat soubory v instalačním místě vývojového prostředí. KEIL verze 4.20 obsahuje zastaralé ovladače, které fungují pouze s procesory řady STM32F10x. Po úpravě knihovny k ovládání ST-LINK programátoru se podařilo KEIL zprovoznit.




Obr. 4.12: Blokové schéma ladícího nástroje od společnosti CodeSourcery

# 5 Realizace

Před samotnou realizací je nutné rozvrhnout blokové schéma zapojení (viz obr. 5.1).




Obr. 5.1: Blokové schéma zapojení autonomního stereo snímače

## 5.1 Univerzální jednotka s procesorem 107/207

### 5.1.1 Požadavky

- Podpora procesorů STM32 řady 107 a 207
- Tlačítko pro resetování
- Přepínač pro volbu náběhu procesoru z RAM nebo FLASH
- Externí krystal pro hlavní hodiny
- Externí krystal pro RTC
- USB konektor pro LOW SPEED
- USB konektor pro HIGH SPEED
- Vyvedení nepoužitých pinů
- Použití lineárního zdroje 3,3V LF33CDT
- Konektor pro CMOS kameru (s využitím periferie z řady 207)
- JTAG konektor pro programování
- Signalizační LED diody napájení a stavů procesoru
- Možnost napájení z USB

### 5.1.2 Návrh elektroniky

Bylo nutné porovnat procesory řady 107 a 207. Odchylky ve vývodech pro pouzdro LQFP100 nalezneme v tabulce 5.1.

| QFP100 | STM32F10X | STM32F20X |
|---|---|---|
| 12 | PD0-OSC_IN | PH0-OSC_IN |
| 13 | PD1-OSC_OUT | PH1-OSC_OUT |
| 19 | VSSA | VDD_12 |
| 20 | VREF- | VSSA |
| 49 | VSS_1 | VCAP_1 |
| 73 | NC | VCAP_2 |
| 74 | VSS_2 | VSS_2 |
| 99 | VSS_3 | VDD_3 |
| 100 | VDD_3 | VDD_SA |

Tab. 5.1: Rozdíl vývodů procesoru STM32F2 a STM32F1

Tyto vývody je nutno navrhnout tak, aby během osazování SMD součástek již byla osazena správná varianta pro daný procesor. Zapojení diskrétních součástek jsem převzal z vývojové jednotky STM3210C-EVAL.

Z pouzdra procesoru byly vyvedeny všechny piny na okraj desky pro snadný přístup k vývodů. Periferie DCMI byla vyvedena na oddělený konektor JP2. Tento konektor má shodné zapojení s kamerovým rozhráním, které bylo dodáno z jiného projektu (schéma zapojení včetně osazení je dostupné jako elektronická příloha práce). Lze proto použít přímý paralelní sběrnicový kabel pro spojení kamery s jednotkou.

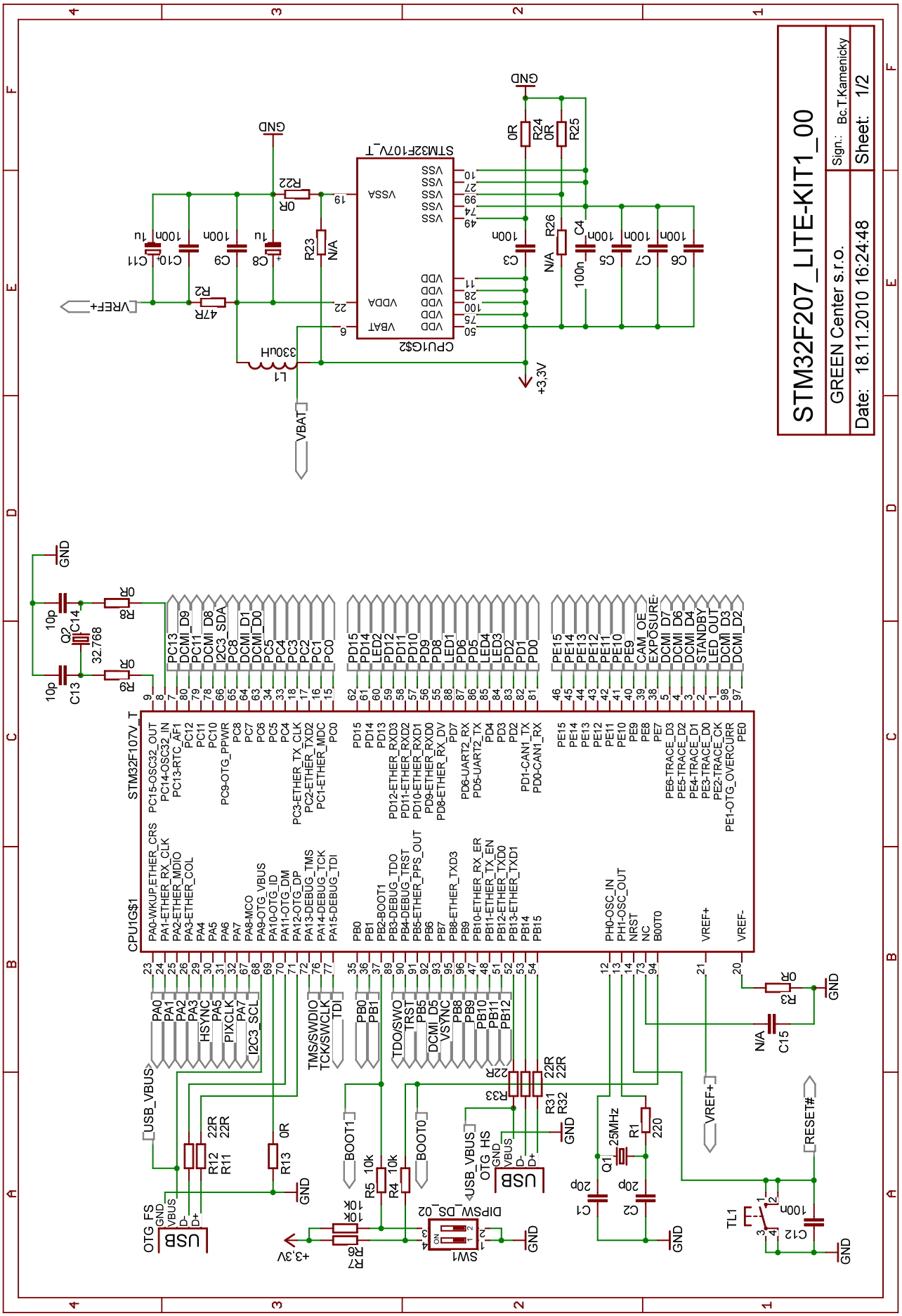
Dále je na jednotce vyveden JTAG konektor pro naprogramovaní a testovaní jednotky. Tento konektor má shodné vyvedení pinů jako vývojová jednotka. Lze tedy použít programovací zařízení ST-Link včetně dodávaného kabelu bez dalších úprav.

Na jednotce nalezneme celkem 5 LED diod. Kde LED diody LD1 až LD4 jsou řízeny procesorem na vývodech shodných s vývojovou jednotkou. Je to z důvodu toho, aby v případě otestování již hotových ukázek pro vývojovou jednotku, fungovaly ukázky i na univerzální jednotce bez úpravy firmwaru. LED dioda LD5 signalizuje stav napájení 3,3 V.

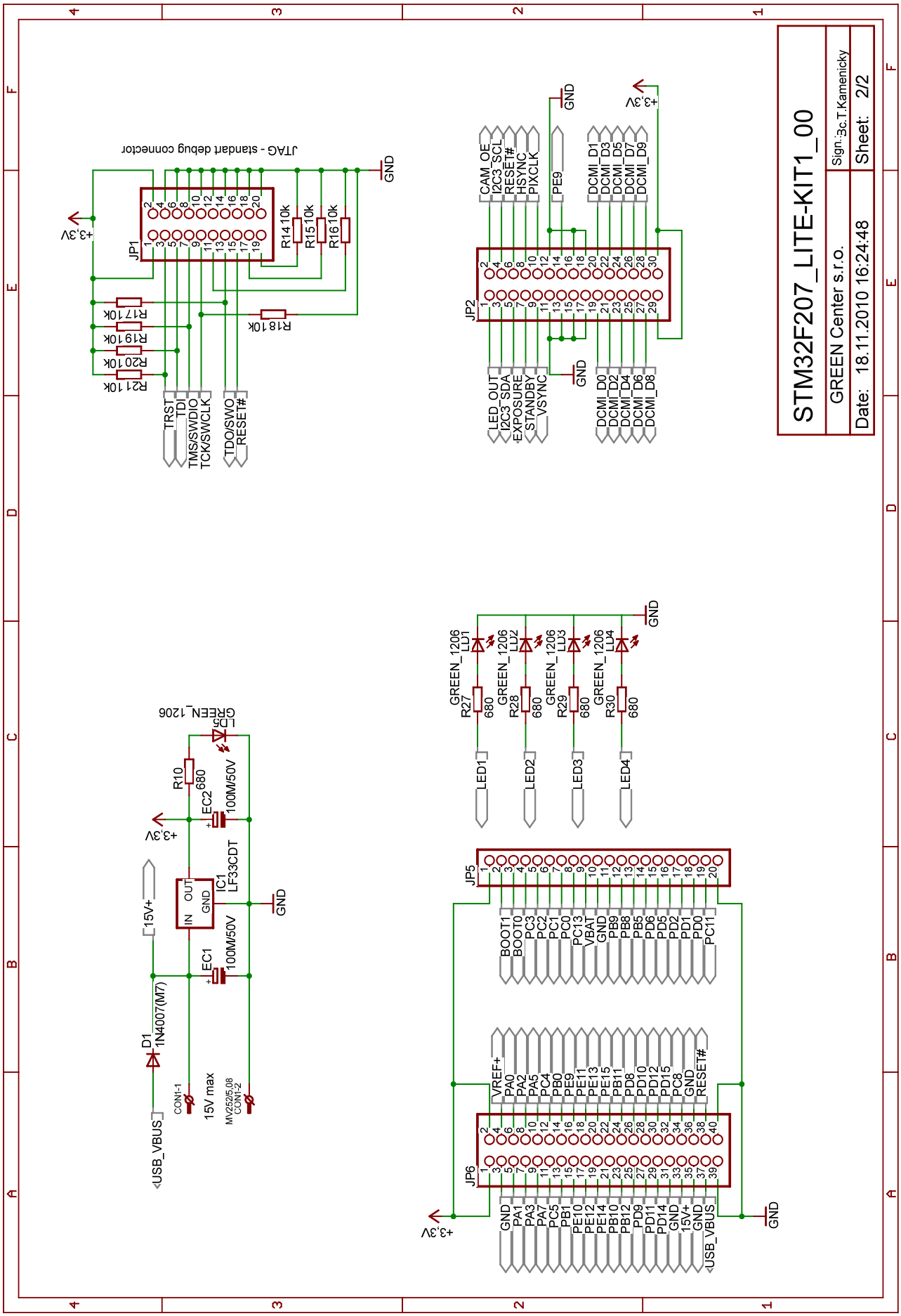
Napájení z USB bylo pro jednoduchost jednotky řešeno pomocí usměrňovací diody 1M4007(M7) připojené do hlavní větve napájení (konektor CON1), kde lze připojit napětí v rozsahu 4,4 V až 15 V. Spodní hodnota napětí je dána lineárním stabilizátorem LF33CDT pro maximální odběr proudu 100mA. Ke vstupům pro napájení procesoru jsou přidány filtrační kondenzátory. Analogová část napájení procesoru je řešena přes filtr s induktorem.

Procesor má k dispozici krystal s kmitočtem 25MHz a s kmitočtem 32.768kHz pro RTC hodiny. Na jednotce je přepínatelný DIP přepínač pro volbu startovací adresy (z RAM nebo FLASH). Na plošném spoji je vytvořen přehledný potisk vývodu s označením a lze také nalézt kalibrační body pro osazovací stroj.

Návrh jednotky byl proveden v návrhovém systému Eagle verze 4.16r. Všechny výrobní podklady včetně seznamu součástek a vazby mezi plošným spojem a schématem zapojení nalezneme jako datovou přílohu této práce.



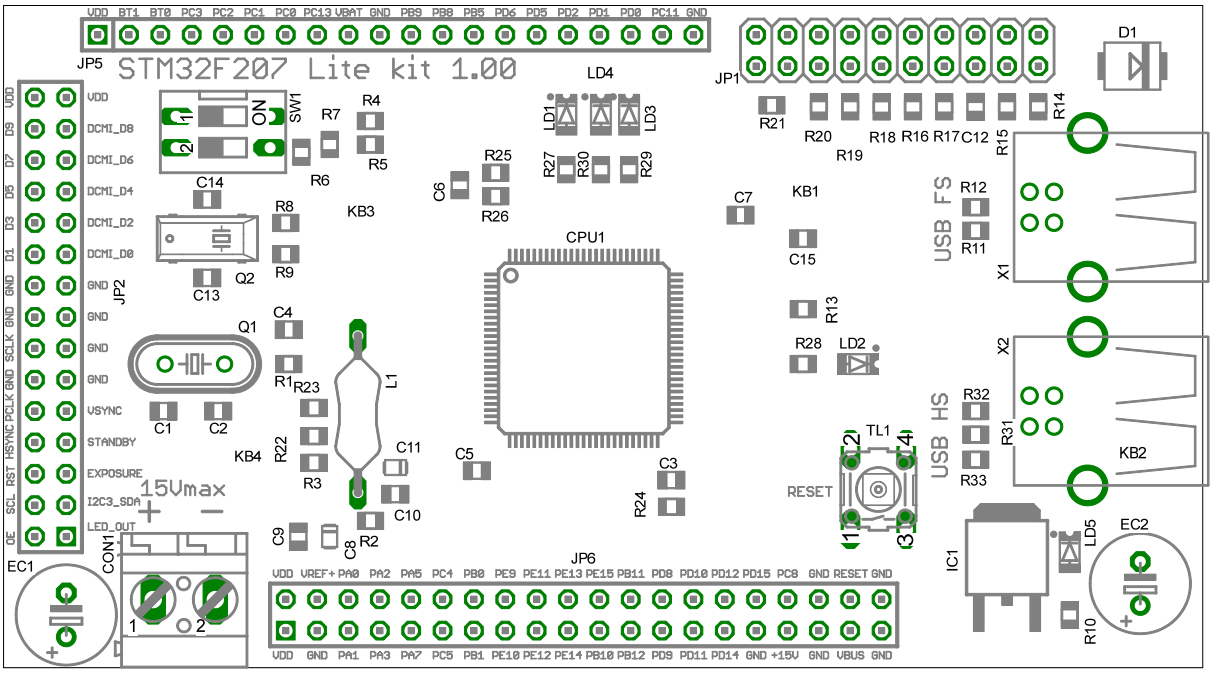

Obr. 5.2: Zapojení univerzální jednotky s procesorem STM32F107/207, schema 1




Obr. 5.3: Zapojení univerzální jednotky s procesorem STM32F107/207, schema 2

| Pozice | Popis | Hodnota | Pouzdro | Ks/deska |
|---|---|---|---|---|
| JP5 | pinhead | | 1X20 | 1 |
| JP1 | pinhead-1 | | 2X10 | 1 |
| JP2 | pinhead | | 2X15 | 1 |
| JP6 | pinhead | | 2X20 | 1 |
| TL1 | tlacitko | | TL6X6 | 1 |
| R3, R8, R9, R13, R22, R24, R25 | odpor | 0R | R0805 | 7 |
| D1 | dioda | 1N4007*M*7 | SMB | 1 |
| C8, C11 | rcl | 1u | R/2012-12R | 2 |
| R4, R5, R6, R7, R14, R15,R16, R17, R18, R19, R20, R21 | odpor | 10k | R0805 | 12 |
| C13, C14 | kond | 10p | C0805 | 2 |
| C1, C2 | kond | 20p | C0805 | 2 |
| R11, R12, R31, R32, R33 | odpor | 22R | R0805 | 5 |
| Q1 | IO-cpu | 25MHz | HC49/S | 1 |
| Q2 | IO-cpu | 32.768 | MM20SS | 1 |
| R2 | odpor | 47R | R0805 | 1 |
| EC1, EC2 | elyt | 100M/50V | RM3,5-8X11MM | 2 |
| C3, C4, C5, C6, C7, C9, C10, C12 | kond | 100n | C0805 | 8 |
| R1 | odpor | 220 | R0805 | 1 |
| L1 | tlumivka | 330uH | EC36 | 1 |
| R10, R27, R28, R29, R30 | odpor | 680 | R0805 | 5 |
| SW1 | prepinac | DIPSW_DS_02 | DS-02 | 1 |
| LD1, LD2, LD3, LD4, LD5 | led | GREEN_1206 | CHIPLED_1206 | 5 |
| IC1 | stabilizatory | LF33CDT | TO252 | 1 |
| CON1 | konektor | MV252/5,08 | MV252/5,08 | 1 |
| C15 | kond | N/A | C0805 | 1 |
| R23, R26 | odpor | N/A | R0805 | 2 |
| CPU1 | IO-cpu | STM32F107V_T | LQFP100 | 1 |
| X1, X2 | con-usb | USB-B-H | USB-B-H | 2 |

Tab. 5.2: Seznam součástek univerzální jednotky s procesorem STM32F107/207




Obr. 5.4: Osazení univerzální jednotky s procesorem STM32F107/207

## 5.2 Základní deska

### 5.2.1 Požadavky

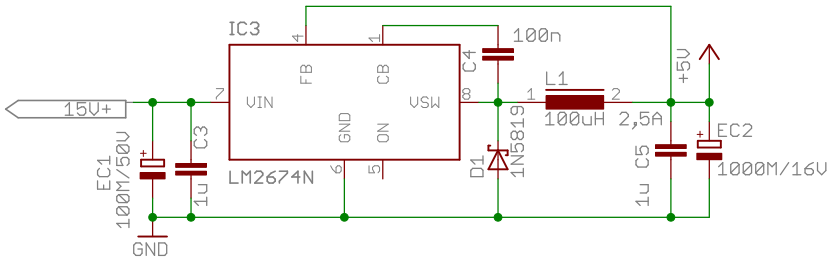
- Slot pro univerzální jednotku
- Piezo měnič pro signalizaci
- Napájecí zdroj pro 5 V
- Napájecí zdroj pro 3,3 V
- Konektor pro CMOS kamery s roztečí 10cm
- Možnost přepínaní kamer včetně PCLK, HSYNC a VSYNC
- RS232 pro výstup dat

### 5.2.2 Návrh elektroniky

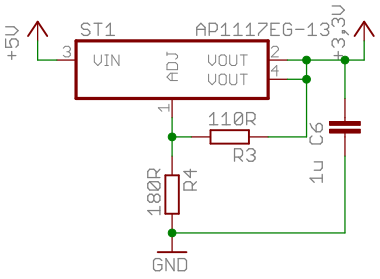
Napájecí zdroj pro 5V (obr. 5.5) byl navržen s obvodem LM2674-5V. Je to spínaný buck regulátor (snižující, step-down). Zapojení včetně hodnot součástek bylo převzato z dokumentace obvodu LM2674. Výstupní zatížitelnost zdroje je 5 V 500 mA.

Napájecí zdroj 3,3 V (obr. 5.6) se skládá z lineárního stabilizátoru AP1117EG-13. Jako referenci používá interní napětí 1,25 V připojené k vývodu Vout. Pro výstupní napětí 3,3 V musí být na vstupu ADJ 2,05 V. Je také nutné aby zpětná vazba byla sestavena z odporu o nízké impedanci. Zaručí to správný chod bez zátěže a přesnou regulaci výstupního napěti.

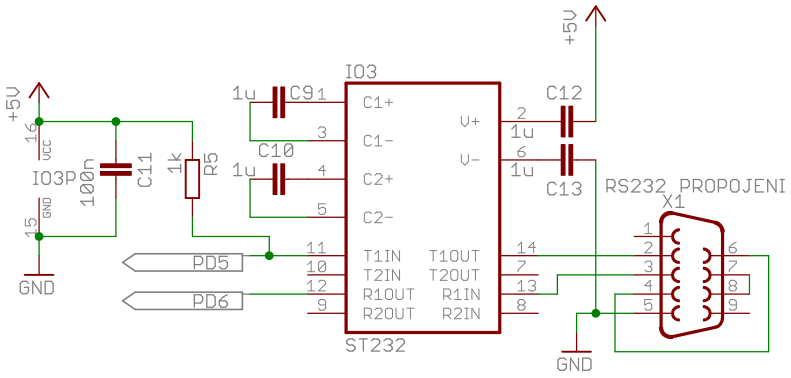
Sériová linka pro UART je řešena pomocí obvodu MAX232 (obr. 5.7). V tomto zapojení je nutné výstup z procesoru ovládat pomoci OD (open drain). Odpor R5 v takovém případě převádí výstup 3,3 V na 5 V logiku.




Obr. 5.5: Zapojení zdroje +5 V

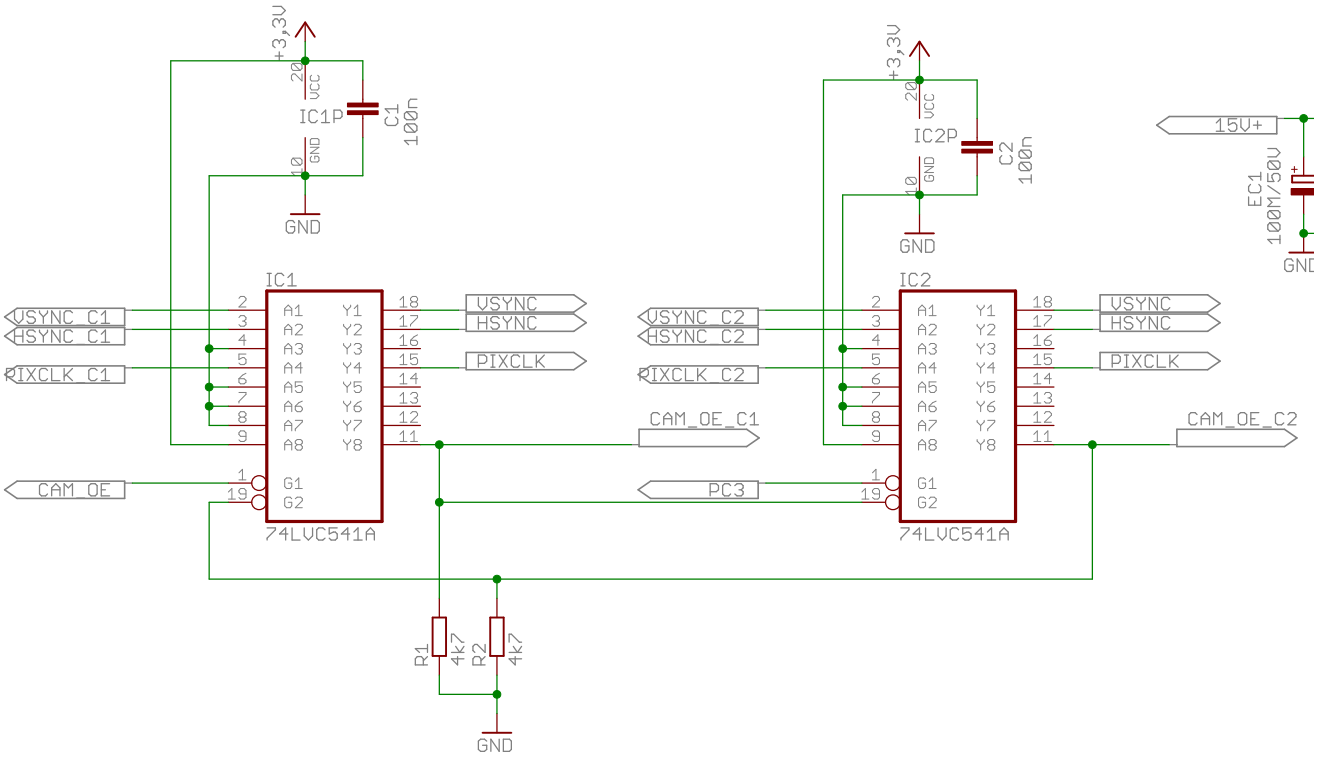



Obr. 5.6: Zapojení zdroje +3,3 V




Obr. 5.7: Zapojení UART linky

Dodané CMOS kamery MT9V032 mají třetí stav pouze na datových pinech. To znemožňuje zapojení kamer na společnou sběrnici. K správnému vyčtení dat jsou nutné řídící signály PCLK (pixel clock), HSYNC (horizontal synchronization) a VSYNC (vertical synchronization). Proto je nutné tyto signály převést na třístavový výstup. K tomu účelu byly použity obvody 74LVC541. Tento obvod je 8 bitový budič sběrnice kompatibilní s 3,3 V logikou. Je také dostatečně rychlý i pro řídící signály s kmitočtem 20MHz (PCLK).




Obr. 5.8: Zapojení budičů 74541

K ovládání výběru kamer je použit signál z obrazového rozhrání CAM_OE a další vývod z univerzální desky PC3 (obr 5.8). Datové výstupy kamer jsou ovládány označeným vodičem CAM_OE_C1 (kamera 1) a CAM_OE_C2 (kamera 2).

Řídící signály z kamer jsou zapojeny jako vstupy 8 bitového budiče. Vzájemná aktivace budičů je vyloučena pomocí vstupu G2. Dokud budič IC1 vysílá, nelze do té doby aktivovat budič IC2 a naopak. To zjednodušuje problém pří náběhu procesoru a ovládání výstupu CAM_OE a PC3. Není nutné dbát na kolizi, která by vznikala bez vzájemného vyloučení.

## 5.3 Oživení přípravku

Z důvodu časového omezení a mé nejistotě funkčnosti budičů sběrnic, jsem obvod vytvořil na jednoduchém pájivém poli. Prvně jsem rozmístil konektory a propojil napájení a země. Poté jsem vytvořil zdroj 5 V a zatížil jsem jej pomocí elektronické zátěže v plném rozsahu proudu 0 až 500 mA. Mezitím jsem měřil zvlnění, které nepřekročilo 100 mV špička-špička. Poté jsem vytvořil 3 V lineární zdroj a opět proměřil pomocí zátěže. Následně jsem vytvořil sériovou linku a postupně oživil kamery. Po úspěšném oživení kamer jsem vytvořil třístavový výstup pro režijní bity senzoru (HSYNC, VSYNC a PCLK). Ověřil jsem funkčnost obvodů 74541 a testoval, zda nedojde k velkému fázovému zpoždění na vodiči PCLK průchodem signálu budičem.

# 6 Firmware a Software

Firmware procesoru byl psán v jazyce C za použití vývojového prostředí KEIL. Software pro ověření funkčnosti kamer byl napsán v jazyce C# ve vývojovém prostředí Microsoft Visual studio express edition.

## 6.1 Firmware

V následujících kapitolách budou objasněny jenom důležité části kódu. Kompletní zdrojový kód je k dispozici jako datová příloha práce.

### 6.1.1 RCC a NVIC

V této kapitole se budeme věnovat nastavení vnitřních sběrnic a povolení přerušení z periferií. RCC je zkratka Reset and Clock control. Je to základní část procesoru, která definuje všechny děličky a násobičky. Dále určuje zdroje vyvolávající restart. Při nastavení kmitočtů sběrnic je nutné dbát na definované maximum podle dokumentace. Procesor jsem nastavil tak, aby jako základní zdroj hodin byl externí krystal na kmitočtu 25MHz. Vnitřní system clock je nastaven na kmitočtu 120MHz, sběrnice APB1 na 30MHz a APB2 na 60MHz. USB periferie je nastavena na kmitočet 48MHz. Ke konfiguraci jsem použil předefinovaná makra, kterým stačí přepsat hodnoty podle násobiček a stačí pak při náběhu jednotky volat funkci *SystemInit*.

Linker překladače je nastaven tak, že jako vstupní funkce není funkce *main* jak je to u PC, ale je to funkce *Reset_Handler*. Tato funkce nastaví všechny globální proměnné a v případě použití statických objektů zavolá jejich konstruktor. Po této operaci se před samotným spuštěním funkce *main* volá právě *SystemInit*, který nastaví rychlost procesoru a jeho sběrnic.

Nyní nám zbývá nastavit NVIC. NVIC je zkratka Nested vectored interrupt controller. Na rozdíl od běžných procesorů, které měly složitou obsluhu přerušení, je procesor s jádrem CORTEX-M3 vybaven tabulkou přerušení. Výhoda je v tom, že když se objeví přerušení, tak se z tabulky načte právě adresa, na kterou přerušení ukazuje. Režie přerušení se tím zkrátí a procesor reaguje na událost mnohem rychleji.

V našem případě budeme potřebovat přerušení od USB a DCMI periferie. Nastavení pro DCMI provedeme následovně:

```
 /* Enable DCMI IRQ Channel */
NVIC_InitStructure.NVIC_IRQChannel = DCMI_IRQn;
NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 0;
NVIC_InitStructure.NVIC_IRQChannelSubPriority = 0;
NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
NVIC_Init(&NVIC_InitStructure);
```

Obdobně je to i pro USB, které navíc nastavíme na nižší prioritu (vyšší číslo znamená nižší prioritu).

```
NVIC_PriorityGroupConfig(NVIC_PriorityGroup_1);

/* Enable the USB Interrupts */
NVIC_InitStructure.NVIC_IRQChannel = OTG_FS_IRQn;
NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 1;
NVIC_InitStructure.NVIC_IRQChannelSubPriority = 0;
NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
NVIC_Init(&NVIC_InitStructure);
```

### 6.1.2 GPIO

Zkratka GPIO znamená General purpose input output. Každý z vývodů procesoru nesoucí tento název lze nastavit do čtyř základních stavů.

- IN, logický vstup
- OUT, logický výstup
- AF, alternate function
- AIN, analogový vstup

Alternate function je vývod obsluhován periferií. Každá periferie má několik možností, kam své vstupy a výstupy přesměruje. To vše se nastavuje na registrech GPIO. U staré řady procesorů STM32F10x lze přesměrovat pouze skupiny vstupů a výstupů. U nové řady lze rozhodovat o každém zvlášť. Dále můžeme nastavit, v případě logického výstupu, rychlost spádové a náběžné hrany. Je to z toho důvodu, že ostré hrany signálu bez správného zakončení vedení, mají značné EMC rušení. Další volba je typ ovládaní výstupu. Zde máme dvě varianty, push-pull (PP), kde výstupy jsou natvrdo ovládány tranzistorem anebo open-drain (OD), kde výstupy jsou pouze přizemněny tranzistorem. Poslední variantou je aktivace rezistorů pull up nebo pull down nebo také obojí. Před každým nastavením registrů je nutné dané registry portu povolit. Ukázka nastavení portu pro řízení LED diod je následující:

```
/* GPIOD clock enable */
RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_GPIOD, ENABLE);
//LED1 is on PD7, LED2 on PD13, LED3 on PD3, LED4 on PD4
GPIO_InitStructure.GPIO_Pin = GPIO_Pin_7 | GPIO_Pin_13 | GPIO_Pin_4 | GPIO_Pin_3;
GPIO_InitStructure.GPIO_Mode = GPIO_Mode_OUT;
GPIO_InitStructure.GPIO_Speed = GPIO_Speed_2MHz;
GPIO_InitStructure.GPIO_OType = GPIO_OType_PP;
GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_NOPULL;

GPIO_Init(GPIOD, &GPIO_InitStructure);
```

Nastavení pro periferii I2C vypadá následovně:

```
//I2C GPIO
/* Configure the I2C GPIOs */
GPIO_InitStructure.GPIO_Pin = GPIO_Pin_9 | GPIO_Pin_8;
GPIO_InitStructure.GPIO_Mode = GPIO_Mode_AF;
GPIO_InitStructure.GPIO_Speed = GPIO_Speed_2MHz;
GPIO_InitStructure.GPIO_OType = GPIO_OType_OD;
GPIO_InitStructure.GPIO_PuPd = GPIO_PuPd_NOPULL;

GPIO_Init(GPIOC, &GPIO_InitStructure);
GPIO_PinAFConfig(GPIOC, GPIO_PinSource9, GPIO_AF_I2C3);
GPIO_PinAFConfig(GPIOC, GPIO_PinSource8, GPIO_AF_I2C3);
```

Obdobným způsobem jsou nastaveny všechny využívané vstupy a výstupy procesoru. Nenastavené vývody jsou implicitně po nastartovaní procesoru nastaveny jako AIN (analogový vstup).

Řízení výstupů pak probíhá zápisem bitů (16 bitů najednou) do příslušného registru GPIO. Od výrobce je dodávaná knihovna, která nejen pomáhá při psaní inicializace, ale také nabízí řízení výstupů.

Pro ovládaní led diody LED1 jsem si vytvořil následující makra:

```
#define LED1_ON()     GPIO_SetBits(GPIOD, GPIO_Pin_7)
#define LED1_OFF()    GPIO_ResetBits(GPIOD, GPIO_Pin_7)
#define LED1_Toggle()   GPIO_ToggleBits(GPIOD, GPIO_Pin_7)
```

### 6.1.3 UART

UART zkratka znamená Universal asynchronous receiver transmitter. Je to sériová linka umožňující přenos 8 bitových dat společně s příznakem pro začátek a konec. Pro jednoduchost nám postačí asynchronní přenos bez hardwarového řízení. Nastavení periferie se provádí následujícím způsobem:

```
  /* Enable USART peripheral clock */
  RCC_APB1PeriphClockCmd(RCC_APB1Periph_USART2, ENABLE);

  USART_InitStructure.USART_BaudRate = 115200;
  USART_InitStructure.USART_WordLength = USART_WordLength_8b;
  USART_InitStructure.USART_StopBits = USART_StopBits_1;
  USART_InitStructure.USART_Parity = USART_Parity_No;
  USART_InitStructure.USART_HardwareFlowControl
  USART_HardwareFlowControl_None;
  USART_InitStructure.USART_Mode = USART_Mode_Rx | USART_Mode_Tx;

  USART_Init(USART2, &USART_InitStructure);

  /* Enable USART */
  USART_Cmd(USART2, ENABLE);
```

Vždy před jakýmkoliv zápisem do registrů je nutné danou periferii zapnout. Takto nastavená linka je pak připravená pro komunikaci. Propojením s PC lze pak v programu hyperterminál jednoduše zapisovat a číst zprávy. V programu je pak definována funkce *fputc*(). Tuto funkci využívá knihovna stdlib. Při volaní funkce *printf* je pak vně volána funkce *fputc*(). Implementace funkce je následující:

```
int fputc(int ch, FILE *f)
{
  while (USART_GetFlagStatus(USART2, USART_FLAG_TC) == RESET) {}

  USART_SendData(USART2, (uint8_t) ch);
  return ch;
}
```

### 6.1.4 I2C

I2C (inter-integrated circuit) je průmyslová sběrnice pro vzájemnou komunikaci hardwaru. Tato sběrnice se skládá z vodičů SDA pro datový přenos a SCL pro hodiny. Spojení je vždy Master-Slave. Ke komunikaci se používá 8 bitová adresace, kde sedmý bit je vždy nastaven na nulu. Po úspěšné adresaci se přenáší data s potvrzením ACK. Po přenosu dat se posílá podmínka STOP. Nastavení pro komunikaci s kamerami je následující:

```
  /* I2C1 clock enable */
  RCC_APB1PeriphClockCmd(RCC_APB1Periph_I2C3, ENABLE);

  /* I2C DeInit */
  I2C_DeInit(I2C3);
```

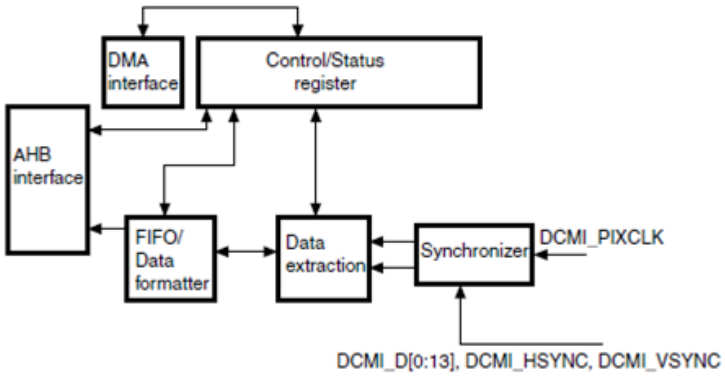
```
/* Enable the I2C peripheral */
I2C_Cmd(I2C3, ENABLE);

/* Set the I2C structure paramaeters */
I2C_InitStruct.I2C_Mode = I2C_Mode_I2C;
I2C_InitStruct.I2C_DutyCycle = I2C_DutyCycle_2;
I2C_InitStruct.I2C_OwnAddress1 = 0x0A;
I2C_InitStruct.I2C_Ack = I2C_Ack_Enable;
I2C_InitStruct.I2C_AcknowledgedAddress = I2C_AcknowledgedAddress_7bit;
I2C_InitStruct.I2C_ClockSpeed = 200000;

/* Initialize the I2C peripheral w/selected parameters */
I2C_Init(I2C3, &I2C_InitStruct);
```

Vlastní adresa 0x0A je v našem případě nepodstatná, neboť kamery jsou pouze v modu slave (pouze naslouchají). Hodiny sběrnice jsou nastaveny na 200kHz. Adresa kamer je nastavena hardwarově na hodnotu 0xB8 a 0xB0. Pro zápis do registrů kamer se pak používá funkce $I2C_Write()$. Ukázka zápisu je následující:

```
I2C_Write(CAM1_ADDR,CAM_Chip_Control,0x0290);
I2C_Write(CAM2_ADDR,CAM_Chip_Control,0x0290);
```

### 6.1.5 DCMI

DCMI je digital camera interface. Tato periferie slouží pro synchronní přenos paralelních dat z obrazového senzoru do paměti procesoru. Periferie používá pro ukládání dat 32 bitový FIFO registr. Je schopna jednotlivé načtené pixely (data) zarovnat a tak požadovat vyčtení jednou za 2, 3 nebo 4 pixely (viz obr. 6.1).




Obr. 6.1: Blokové schéma zapojení DCMI procesoru STM32F207

Podporuje také DMA přenos. Na periferii můžeme nastavit polaritu vstupních hodin, polaritu vertikálního a horizontálního řízení. Můžeme také nastavit šířku přijímaných dat. Systémové hodiny pro funkčnost kamer musíme generovat zvlášť pomocí čítačů v procesoru. Nastavení DCMI periferie provedeme následovně:

```
RCC_APB2PeriphClockCmd(RCC_APB2Periph_SYSCFG, ENABLE);

/* Enable DCMI clock */
RCC_AHB2PeriphClockCmd(RCC_AHB2Periph_DCMI, ENABLE);

/* DCMI configuration */
DCMI_InitStructure.DCMI_CaptureMode = DCMI_CaptureMode_SnapShot;
DCMI_InitStructure.DCMI_SynchroMode = DCMI_SynchroMode_Hardware;
```

```
DCMI_InitStructure.DCMI_PCKPolarity = DCMI_PCKPolarity_Rising;
DCMI_InitStructure.DCMI_VSPolarity = DCMI_VSPolarity_Low;
DCMI_InitStructure.DCMI_HSPolarity = DCMI_HSPolarity_Low;
DCMI_InitStructure.DCMI_CaptureRate = DCMI_CaptureRate_All_Frame;
DCMI_InitStructure.DCMI_ExtendedDataMode = DCMI_ExtendedDataMode_8b;

DCMI_Init(&DCMI_InitStructure);
```

Lze také nastavit nepřetržité mód snímkování, nebo jen po snímcích a nebo také vynechávání snímků. Je důležité vědět to, že po načtení jednoho snímku se zavolá přerušení $DCMI_FRAME_CompleteIRQ($
V této rutině se provede přepnutí budičů na druhou kameru a proces snímku se opakuje, tak jak je to vidět z následujícího kódu:

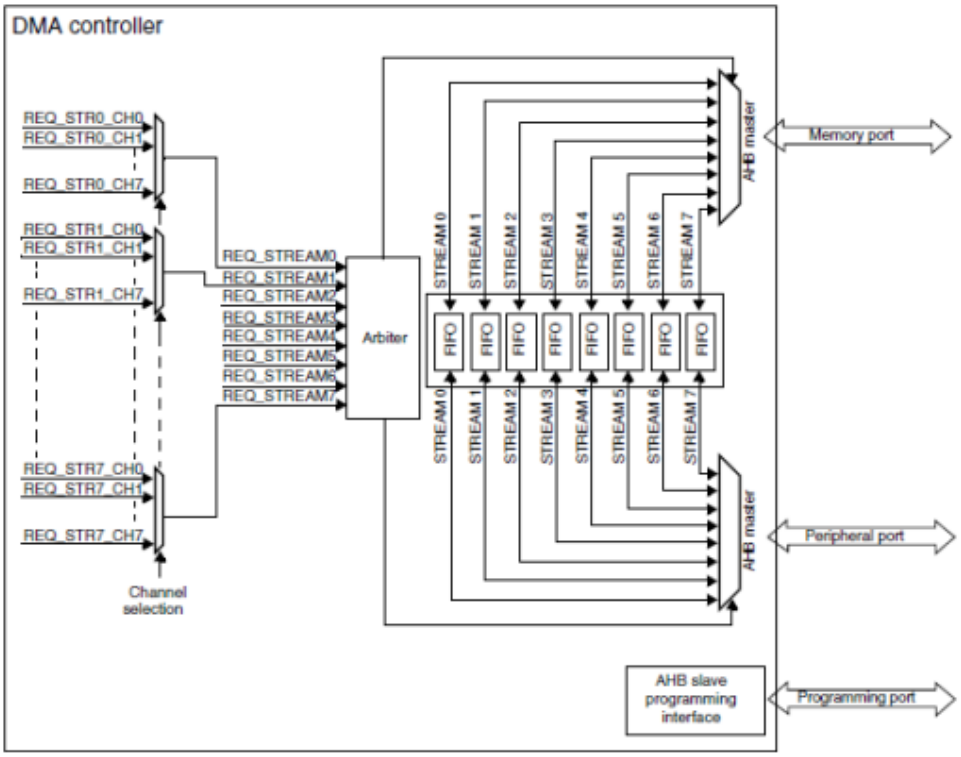
```
if (switch_cams) {
        CAM2_OE_OFF();
        CAM1_OE_ON();
        switch_cams=0;
    } else {
        CAM1_OE_OFF();
        CAM2_OE_ON();
        switch_cams=1;
    }
start_exposure=1;

while(CAM_CHECK_VSYNC()||CAM_CHECK_HSYNC()) {}

 /* Start Image capture */
 DCMI_CaptureCmd(ENABLE);
```

Funkce *while* zde pozdrží procesor po dobu, než CMOS senzor nepřejde do definovaného stavu. Až poté se aktivuje nové vyčtení snímků.

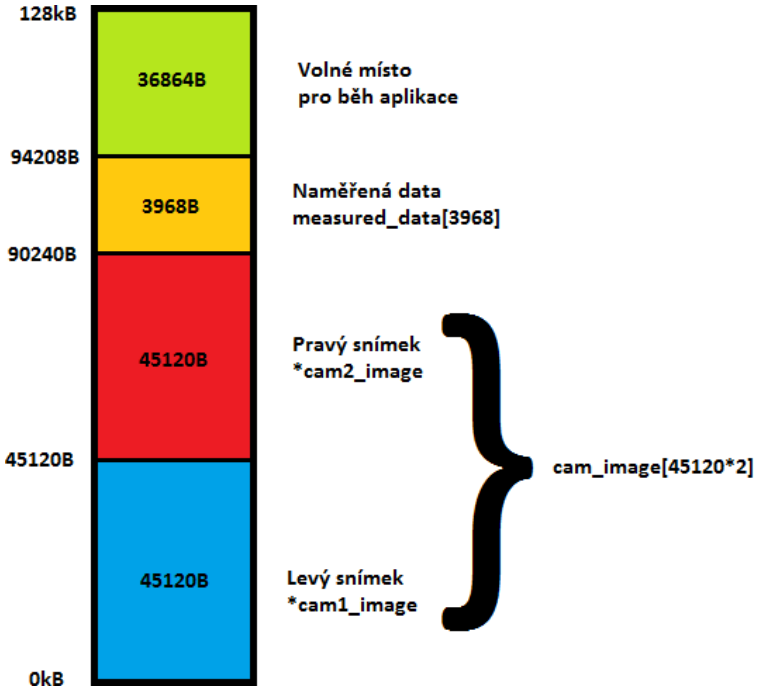
### 6.1.6 DMA

DMA je zkratka Direct Memory Access. Je to periferie používaná pro přenos dat mezi jinou periferii a pamětí, nebo pamětí a pamětí, nebo periférií a periferií bez použití jádra procesoru. Skládá se z 8 proudů (stream), kde každý proud může být naprogramován samostatně (viz obr. 6.2):

DMA přenos využijeme pro přenos dat mezi DCMI a pamětí RAM. Předtím je nutné ale rozvrhnout, do které části paměti RAM se data budou ukládat. Z dokumentace pro obrazový senzor jsem vyčetl, že lze redukovat obraz tak, aby jeden snímek mněl rozlišení 752x60 pixelů. Toho jsem docílil tak, že jsem nastavil okénkování (aktívní oblast) na 752x240 a vyčtení každého čtvrtého řádku. Tímto způsobem docílíme velikost jednoho snímku o hodnotě 45120bajtů. Máme celkem k dispozici 128kB paměti RAM. Z toho plyne, že do paměti se nám vejde jak levý tak i pravý snímek z kamer a ještě nám zbyde 40832 bajtů pro samotnou funkčnost programu. Rozvržení paměti RAM je na obrázku 6.3.




Obr. 6.2: Blokové schéma zapojení DMA procesoru STM32F207




Obr. 6.3: Softwarové přidělení paměti v programu pro autonomní stereo símač

Samotné nastavení DMA periferie je následující:

```
/* Enable DMA2 clock */
RCC_AHB1PeriphClockCmd(RCC_AHB1Periph_DMA2, ENABLE);

/* DMA2 Stream1 Configuration */
DMA_DeInit(DMA2_Stream1);

DMA_InitStructure.DMA_Channel = DMA_Channel_1;
DMA_InitStructure.DMA_PeripheralBaseAddr = (uint32_t)DCMI_DR_ADDRESS;
DMA_InitStructure.DMA_Memory0BaseAddr = (uint32_t)&cam_image;
DMA_InitStructure.DMA_DIR = DMA_DIR_PeripheralToMemory;
DMA_InitStructure.DMA_BufferSize = (pixelu)/2; //Prenasime po 4 bajtech
DMA_InitStructure.DMA_PeripheralInc = DMA_PeripheralInc_Disable;
DMA_InitStructure.DMA_MemoryInc = DMA_MemoryInc_Enable;
DMA_InitStructure.DMA_PeripheralDataSize = DMA_PeripheralDataSize_Word;
DMA_InitStructure.DMA_MemoryDataSize = DMA_MemoryDataSize_Word;
DMA_InitStructure.DMA_Mode = DMA_Mode_Circular;
DMA_InitStructure.DMA_Priority = DMA_Priority_High;

DMA_InitStructure.DMA_FIFOMode = DMA_FIFOMode_Disable;
DMA_InitStructure.DMA_FIFOThreshold = DMA_FIFOThreshold_Full;
DMA_InitStructure.DMA_MemoryBurst = DMA_MemoryBurst_Single;
DMA_InitStructure.DMA_PeripheralBurst = DMA_PeripheralBurst_Single;

DMA_Init(DMA2_Stream1, &DMA_InitStructure);
```

Z nastavení DMA je patrné, že přenosem jednoho snímku je vnitřní ukazatel DMA na adrese $cam2\_image$. Proto po každém půl přenosu (half-transfer) přepínám kamery. Tím jsem docílil nepřetržitý update paměti podle snímků z kamer. Po dvou snímcích je ukazatel automaticky přesunut na začátek $cam1\_image$.

### 6.1.7 TIMER1

TIMER1 je vnitřní čítač, který na základě své konfigurace může na svém výstupu generovat různé kmitočty a střídy. V mém případě jsem jako vstup čítače nastavil vnitřní hodiny na kmitočtu 120MHz a jako výstup jsem nastavil přepínání výstupu při přetečení čítače přes hodnotu 2. Tím je na výstupu generován hodinový signál o kmitočtu 20MHz s 50% střídou. Tento signál je použít jako SCLK pro senzory. Nastavení čítače v kódu:

```
/* TIM1 clock enable */
RCC_APB2PeriphClockCmd(RCC_APB2Periph_TIM1, ENABLE);

TIM_TimeBaseStructure.TIM_Prescaler = 0;
TIM_TimeBaseStructure.TIM_CounterMode = TIM_CounterMode_Up;
TIM_TimeBaseStructure.TIM_Period = 2;
TIM_TimeBaseStructure.TIM_ClockDivision = 0;
TIM_TimeBaseStructure.TIM_RepetitionCounter = 3;

TIM_TimeBaseInit(TIM1, &TIM_TimeBaseStructure);

  /* Channel 1 Configuration in PWM mode */
TIM_OCInitStructure.TIM_OCMode = TIM_OCMode_Toggle;
TIM_OCInitStructure.TIM_OutputState = TIM_OutputState_Enable;
TIM_OCInitStructure.TIM_OutputNState = TIM_OutputNState_Disable;
TIM_OCInitStructure.TIM_Pulse = 0;
TIM_OCInitStructure.TIM_OCPolarity = TIM_OCPolarity_Low;
TIM_OCInitStructure.TIM_OCNPolarity = TIM_OCNPolarity_Low;
TIM_OCInitStructure.TIM_OCIdleState = TIM_OCIdleState_Set;
TIM_OCInitStructure.TIM_OCNIdleState = TIM_OCIdleState_Reset;

TIM_OC1Init(TIM1, &TIM_OCInitStructure);
```

```
  TIM_OC1PreloadConfig(TIM1, TIM_OCPreload_Disable);

  /* Automatic Output enable, Break, dead time and lock configuration*/
  TIM_BDTRInitStructure.TIM_OSSRState = TIM_OSSRState_Enable;
  TIM_BDTRInitStructure.TIM_OSSIState = TIM_OSSIState_Enable;
  TIM_BDTRInitStructure.TIM_LOCKLevel = TIM_LOCKLevel_OFF;
  TIM_BDTRInitStructure.TIM_DeadTime = 0;
  TIM_BDTRInitStructure.TIM_Break = TIM_Break_Disable;
  TIM_BDTRInitStructure.TIM_BreakPolarity = TIM_BreakPolarity_High;
  TIM_BDTRInitStructure.TIM_AutomaticOutput = TIM_AutomaticOutput_Enable;

  TIM_BDTRConfig(TIM1, &TIM_BDTRInitStructure);

  TIM_SelectOutputTrigger(TIM1, TIM_TRGOSource_OC1Ref);

  /* TIM1 counter enable */
  TIM_Cmd(TIM1, ENABLE);
```

### 6.1.8 USB

USB znamená Universal Serial Bus. Tato periferie má komplexní nastavení a je dobře podporována výrobcem. Proto jsem vycházel z příkladu, který vytvoří převodník UART na USB. V PC, po připojení jednotky se zahlásí virtuální COM port od firmy ST. Celou část kódu, která obsluhuje USB port, jsem převzal z tohoto příkladu. Dovolilo mi to tak snadnou implementaci USB rozhrání, které jsem na poslední aplikační vrstvě upravil, pro účely stereo snímače.

Nebudu zde popisovat inicializaci USB periferie, protože by zabrala značnou část dokumentu. Zaměřím se na funkce, které se volají při příjmu a odesílaní dat. V případě, že po virtuálním COM portu dorazí data se po složité operaci protokolu USB, zavolá funkce *USB_To_CPU_Send_Data*().

```
 void USB_To_CPU_Send_Data(uint8_t * USB_Rx_Buffer, uint16_t USB_Rx_Cnt)
{
     LED3_Toggle();

     if (USB_Rx_Cnt>0)
     {
         if (USB_Rx_Buffer[0]=='D')
             {
                 if(USB_Tx_State==0)
                 {
                 USB_Tx_State=2;
                 USB_Buffer=&(cam_image[0]);
                 MEMORY_Rx_ptr_out=0;
                 MEMORY_Rx_ptr_in=pixelu*pocet_kamer;
                 USB_Tx_State=0;
                 }
             } else
         if (USB_Rx_Buffer[0]=='E')
             {
                 if(USB_Tx_State==0)
                 {
                 USB_Tx_State=2;
                 USB_Buffer=&(cam_image[pixelu]);
                 MEMORY_Rx_ptr_out=0;
                 MEMORY_Rx_ptr_in=pixelu;
                 USB_Tx_State=0;
                 }
             } else
         if (USB_Rx_Buffer[0]=='F')
             {
                 if(USB_Tx_State==0)
                 {
                 USB_Tx_State=2;
```

```
                USB_Buffer=&(measured_data[0]);
                MEMORY_Rx_ptr_out=0;
                MEMORY_Rx_ptr_in=3968;
                USB_Tx_State=0;
                }
            }
        }
}
```

V této funkci detekuji v přijímaných datech znak $D$, $E$ anebo $F$. V případě že USB knihovna nevysílá data, je daný znak zpracován. V případě příjmu znaku $D$ je adresa vysílacího ukazatele nastavena na počátek *cam_image*. Velikost dat je nastavena na dva snímky (90240 bajtů). Od tohoto okamžiku se čeká na zavolaní funkce:

```
void Handle_USBAsynchXfer (void)
```

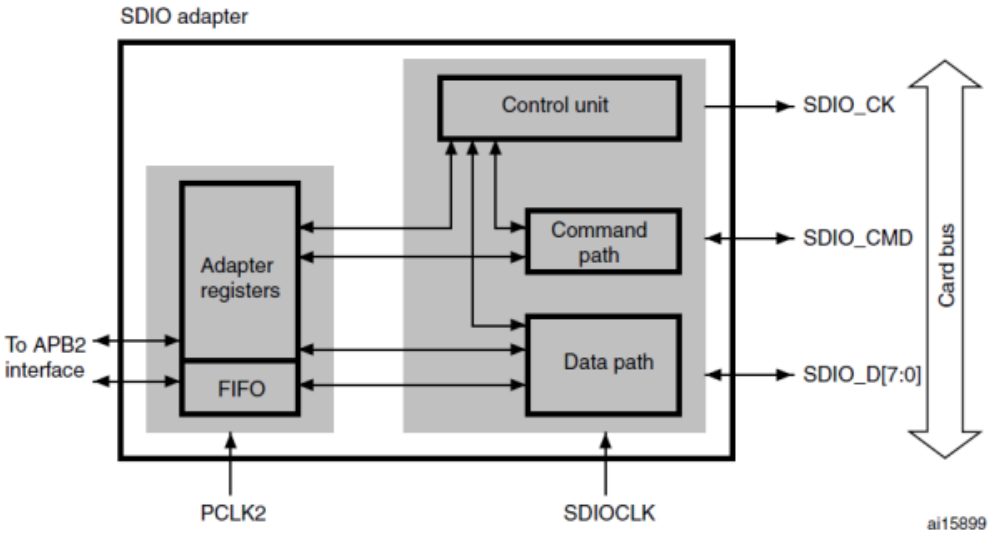
Tato funkce se volá v okamžiku, kdy dojde k přenosu nového rámce (frame). Funkce zkontroluje stav ukazatelů a v případě, že jsou dostupná nová data, začne přenos prvního paketu. Tento přenos nastartuje automatizovaný přenos dalších paketů, dokud nedojde k přenosu všech nastavených dat. Funkce, která se volá pro automatický přenos je:

```
void EP1_IN_Callback (void)
```

Obdobně je obsluhováno přijmutí znaku $E$ a $F$. Znak $E$ vrátí první obraz a znak $F$ naměřená data. Je nutné upozornit, že ovladač pro PC je omezen při příjmu dat. Při načítaní dat z ovladače je nutno ze strany jednotky naplnit zásobník na 4kB. Až poté se data objeví v aplikaci. Z tohoto důvodu jsou data pro naměřené hodnoty o velikosti 3968 bajtů, aby při vyčtení snímků a následném vyčtení naměřených dat, došlo k úplnému vyprázdnění zásobníku na straně ovladače pro PC.

### 6.1.9 SDIO

SDIO je periferie obsluhující externí paměťové medium SD memory card nebo MMC (Multi Media Card).




Obr. 6.4: Blokové schéma zapojení SDIO procesoru STM32F207

Táto periferie zahrnuje kompletní protokol pro řízení paměťových médií. Knihovna dodávaná od výrobce obsahuje řízení periferie. K této knihovně je nutné importovat knihovnu pro souborový systém FAT32. Vrstvu souborového systému je pak nutné spojit s vrstvou pro souborový přístup knihovny libc z knihovny newlib. Jako vhodná knihovna pro souborový systém je knihovna efsl.

Pro správnou funkci knihovny SDIO je nutné povolit přerušení následujícím způsobem:

```
/* Configure the NVIC Preemption Priority Bits */
NVIC_PriorityGroupConfig(NVIC_PriorityGroup_1);

NVIC_InitStructure.NVIC_IRQChannel = SDIO_IRQn;
NVIC_InitStructure.NVIC_IRQChannelPreemptionPriority = 0;
NVIC_InitStructure.NVIC_IRQChannelSubPriority = 0;
NVIC_InitStructure.NVIC_IRQChannelCmd = ENABLE;
NVIC_Init(&NVIC_InitStructure);
```

Po této operaci se může zavolat z knihovny SDIO funkce $SD\_Init()$. Tato funkce nastaví všechny potřebné registry pro správnou funkci s SD kartou. Poté jsou z knihovny dostupné funkce:

```
SD_Error SD_ReadBlock (uint8_t *readbuff,   uint32_t ReadAddr, uint16_t BlockSize);
SD_Error SD_WriteBlock(uint8_t *writebuff, uint32_t WriteAddr, uint16_t BlockSize);
```

### 6.1.10 Ethernet

Ethernet je periferie pro komunikaci po počítačové sítí. K její funkčnosti je nutná fyzická vrstva s rozhráním MII (Media independend interface) anebo RMII (Reduced media independend interface). Periferie zároveň podporuje standart IEEE 1588-2002 (PTP) pro přesnou časovou synchronizaci. Tento standart je nutný v okamžiku použití vícero jednotek stereo snímače na větší vzdálenost. Spojením s SD kartou dostaneme funkci pro synchronní snímkování. To je vhodné pro detekci vzdálených překážek, kde snímky se shodujícím časovým razítkem lze zpracovat a rozhodnou o její vzdálené pozici.

Pro implementaci PTP protokolu, který je částečně řešen v knihovnách pro Ethernet, nezbyl čas. Důvodem je i fakt že při použití DCMI periferie na pouzdru LQFP100 lze použít pouze RMII. RMII má tu nevýhodu že pracuje na kmitočtu až 50MHz. Z tohoto důvodu je nutný nový návrh procesorové desky včetně fyzické vrstvy pro Ethernet. Dále je pak nutné mít dvě jednotky stereo snímače pro ověření funkčnosti protokolu PTP.

### 6.1.11 Nastavení kamer

CMOS snímače se nastavují pomocí zápisu do registrů přes sběrnici I2C. První zápis způsobí restart snímačů. Poté se nastaví registr Chip control, ve kterém definujeme počet řádků, které se mají přeskočit. Následně se definují rozměry okénkování a jednorázové snímkování pomocí vstupu vývodu exposure. Posledně se zakáže funkce automatic exposure control. Je to z toho důvodu, aby jsme si byly jisti, že se kamery nerozladí při automatické kalibraci. Doba expozice je řízená programem následovně:

```
if (prumer_jasu>140)
{
  //sniz zesileni nebo dobu integrace
  if (exposure_time>1)
  {
```

```
    exposure_time--;
    I2C_Write(CAM1_ADDR,CAM_Total_Shutter_Width, exposure_time);
    I2C_Write(CAM2_ADDR,CAM_Total_Shutter_Width, exposure_time);
    }

  }
  else if (prumer_jasu<120)
  {
    //Zvis zesileni nebo dobu integrace
    if (exposure_time<32766)
    {
    exposure_time++;
    I2C_Write(CAM1_ADDR,CAM_Total_Shutter_Width, exposure_time);
    I2C_Write(CAM2_ADDR,CAM_Total_Shutter_Width, exposure_time);
    }
```

Po ustálené době bude prúmerný jas, který je měřen z prostředního řádku, nastaven v rozmezí hodnot 120 až 140.

### 6.1.12 Implementace metody sumace

Jádrem programu je nekonečná smyčka, uvnitř které je cyklus, který prochází snímek iterací řádku i a sloupce j. Vyčtení pixelů z paměti se provádí následujícím způsobem:

```
big=j+i*752;
pixel_L=cam_image[big];
pixel_R=cam_image[45120+big];
```

Poté se spouští algoritmus metody sumace:

```
if (pixel_L>pixel_R)
        sum+=pixel_L-pixel_R;
        else sum+=pixel_R-pixel_L;
```

Po ukončení cyklu se hodnota sumace uloží do předem definované pozice naměřených dat:

```
*measured_sum=sum;
sum=0;
```

A celý proces se znovu opakuje. V rámci detekce polohy je implementován na prostředním řádku jednoduchý prahový detektor. Prahovací úroveň je adaptivní a je vypočtena z předchozího řádku pomocí průměrování.

### 6.1.13 Implementace metody lokální disparity obrazů

Lokální disparita je implementována o něco složitějším algoritmem než metoda sumace. Před samotným detektorem hran je obraz diference filtrován IIR filtrem 1. řádu se zlomem 0,5 normované prostorové frekvence.

Samotný detektor je naprogramován jako stavový automat. Nejdříve se očekává záporný pulz a jeho hrana. V případě nalezení se uloží aktuální sloupec do proměnné *horizontal*[0]. Následně se očekává spádová hrana záporného pulzu, při které se porovnává, zda hrana je zapříčiněna levým nebo pravým obrazem (viz teorii metody disparity obrazů). Opět se poloha hrany zaznamená do proměnné *horizontal*. Poté se detekuje kladný pulz a jeho hrany. V posledním stavu označeným číslem 3 se provede test, zda vypočtena vzdálenost je větší než definovaná hodnota a výsledek

měření se zapíše do tabulky *measured_vertical* vždy pro levý a pravý obraz zvlášť. Nakonec se porovná výsledná hodnota disparity, která se zapíše v případě, že jde o nejbližší překážku.

Samotná implementace algoritmu je následující:

```
case 0:
if ((diference_pixelu_IIR)<(-diff_prah))  {
        der=1;
        horizontal[1]=j; }
break;
case 1:
    if ((diference_pixelu_IIR-histereza)>(-diff_prah)) {
      if (pixel_L>prumer_jasu) {
     der=4;
         horizontal[2]=j;
         tloustka_prekazky_L=horizontal[2]-horizontal[1]; }
else {
            der=2;
        horizontal[3]=j; }
        }
break;
case 2:
    if (diference_pixelu_IIR>diff_prah) {
        der=3;
        horizontal[2]=j;
        tloustka_prekazky_L=horizontal[2]-horizontal[1]; }
break;
case 3:
    if ((diference_pixelu_IIR+histereza)<diff_prah) {
        der=0;
        horizontal[4]=j;

        distance=(horizontal[3]-horizontal[1]);
        distance=distance+(horizontal[4]-horizontal[2]);

        if (distance>treshold) {
            while (USB_Tx_State!=0);
            measured_vertical[vertical_index]=horizontal[1]-2;
            measured_vertical[vertical_index+1]=horizontal[2]-2;
            measured_vertical[20+vertical_index]=horizontal[3]-2;
            measured_vertical[21+vertical_index]=horizontal[4]-2;
            vertical_index=vertical_index+2;
            if (vzdalenost_prekazky_max<distance)
                vzdalenost_prekazky_max=distance;
            }
        else if  ((tloustka_prekazky_L+tolerancia_prekazky)<(j-horizontal[3]))
                {
                    der=0;
                }
        }
break;
case 4:
     if (diference_pixelu_IIR>diff_prah) {
        der=3;
        horizontal[3]=j; }
break;
```

V naměřených datech *measured_vertical* máme informaci o horizontální poloze překážek. V proměnné *vzdalenost_prekazky_max* se nachází vzdálenost nejbližší překážky. Po ukončení cyklu se výsledek vzdálenosti uloží do proměnné *measured_sum*.

### 6.1.14 Souhrn

Program periodicky načítá snímky z levé a pravé kamery a pomocí DMA je ukládá do paměti RAM nezávisle na běžícím algoritmu, který s těmito hodnotami pracuje. Připojením na PC

pomocí USB portu se v PC vytvoří virtuální COM port. Zasláním znaku *D* na port obdržíme surová data levého a pravého obrázku. Zasláním znaku *F* obdržíme naměřená data (hrany a vzdálenost nejbližší překážky). Výběr měřící metody se provádí zakomentováním jednoho ze dvou následujících řádků:

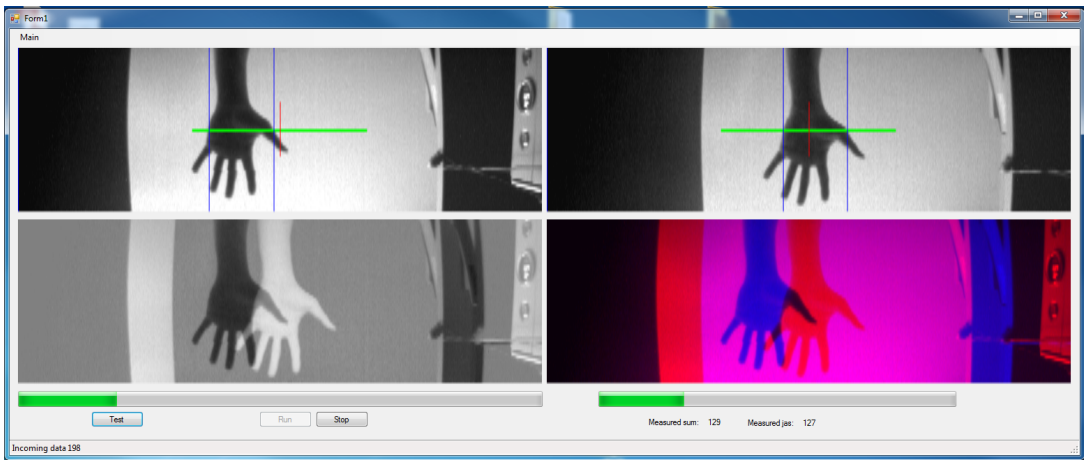
```
#define METODA_MERENI_1     1
#define METODA_MERENI_2     1
```

Nelze provozovat obě měření současně, nakolik využívají stejný pracovní prostor proměnných.

## 6.2 Software

Software ImageGrabber byl vyvíjen současně s programem pro jednotku stereo snímače.

### 6.2.1 GUI




Obr. 6.5: Uživatelské rozhrání programu ImageGrabber

ImageGrabber má v celku jednoduché ovládání. Tlačítkem Test provedeme test zobrazení obrázku GUI aplikace. Tlačítkem Stop zastavíme přenos dat mezi jednotkou stereo snímače a počítačem. Tlačítkem Run spustíme přenos dat a automatickou vizualizaci dat z jednotky. První kvadrant obrázku je pohled z pravé kamery roztažen do rozlišení 752x240 (z kamer se vyčítá obraz o rozlišení 752x60, který je deformován vlivem vyčítání každého čtvrtého řádku). Druhý kvadrant ukazuje pohled z levé kamery. Třetí kvadrant je vizualizace dat, se kterými jednotka pracuje při hledání hran. Čtvrtý kvadrant je poskládáním levého a pravého obrazu do jednoho pomocí dvou barev, kde levá kamera nese barvu červenou a pravá kamera barvu modrou. Levý bar graf značí aktivitu přenosu. Pokud se nehýbe, program nejspíš zůstal někde viset nebo je zacyklen. Pravý bar graf vizualizuje naměřenou vzdálenost. Modré vertikály prvého a druhého kvadrantu jsou vizualizována data *measured_vertical* z jednotky (určená v procesoru STM32).

### 6.2.2 Inicializace sériové linky

K inicializaci sériové linky používám knihovnu System.IO.Ports. Program očekává jednotku na portu COM12. Přenosovou rychlost není potřeba definovat, protože port je pouze virtuální. Po

stisknutí tlačítka Run se program nejdříve pokusí port COM12 otevřít. Pokud otevření selže, pokusí se jej zavřít a opět otevřít. Algoritmus pro otevření portu je následující:

```
IVSC(() => toolStripStatusLabel1.Text = "Connecting");

try
   {
     _serialPort.DiscardInBuffer();
     _serialPort.DiscardOutBuffer();
     _serialPort.Dispose();
     _serialPort.Close();
    }
    catch { }

System.Threading.Thread.Sleep(100);

 _serialPort.InitializeLifetimeService();

try
    {
      _serialPort.Open();
       System.Threading.Thread.Sleep(100);
      _serialPort.ReadExisting();
     }
    catch (Exception exc)//System.IO.IOException
    {
       bool succes = false;
       int chances = 1000;

       while (!succes)
        {
          succes = true;
          chances--;
          try
              {
               _serialPort.Close();
              }
             catch { };

          System.Threading.Thread.Sleep(10);

           try
                {
                  _serialPort.Open();
                }
                catch { succes = false; };

           System.Threading.Thread.Sleep(10);
           if ((!succes) && (chances < 0))
              {
                IVSC(() => toolStripStatusLabel1.Text = "Failed to open the port!");
                thread_run = false;
                return;
              }
           }
        }

IVSC(() => toolStripStatusLabel1.Text = "Connected!");
```

Po připojení se spustí nové vlákno *DataHarvester*(), které zasílá znaky $D$ a $F$ do virtuální linky a vyčítává data zasílané z jednotky.

### 6.2.3 Vyčtení dat

Z důvodu omezení ovladače, který nepředá přijímané data z jednotky dříve, než se 4kB zásobník naplní, jsou data kompletována pro jeden přenos tak, aby byla celočíselným násobkem 4096

bajtů. *DataHarvester*() provede zápis znaku *D*, viz následující kód:

```
_serialPort.Write(byte_sync_get_image, 0, 1);
```

Vyprázdní paměť pro nové snímky a naměřené data a poté zkontroluje zda jsou data k načtení.

```
for (int j = 0; j < (752 * 60); j++) image1_data[j] = 0;
           for (int j = 0; j < (752 * 60); j++) image2_data[j] = 0;
           for (int j = 0; j < (1024); j++) measure_data[j] = 0;
           System.Threading.Thread.Sleep(10);

 if (_serialPort.BytesToRead != 0)
                {...process image....}
```

V případě že žádná data nejsou k dispozici, je vypsána hláška „No data on line“. V případě že data jsou k dispozici, vyčítá se pouze celistvý násobek 4096 bajtů. Načtené snímky a výsledky algoritmů nalezneme v proměnných *image*1_*data*, *image*2_*data* a *measure*_*data*.

### 6.2.4 Zpracování dat

Po načtení dat dochází k jejich zpracování. Data snímků jsou procházena pixel po pixelu a jsou transformována do bitmapy, která je pak zobrazena v programu pomocí rozhrání GUI.

```
for (current_line = 0; current_line < vyska; current_line++) {
            for (pixel_in_line = 0; pixel_in_line < sirka; pixel_in_line++) {
              big = sirka * current_line + pixel_in_line;

              pixel_cam1 = image1_data[big];
              pixel_cam2 = image2_data[big];

              diff_pixel = 128+((pixel_cam1 - pixel_cam2)/2);

              cam1_bitmap.SetPixel(pixel_in_line, current_line,
                        Color.FromArgb(pixel_cam1, pixel_cam1, pixel_cam1));
              cam2_bitmap.SetPixel(pixel_in_line, current_line,
                        Color.FromArgb(pixel_cam2, pixel_cam2, pixel_cam2));

              diff_bitmap.SetPixel(pixel_in_line, current_line,
                        Color.FromArgb(diff_pixel, diff_pixel, diff_pixel));

              mix_bitmap.SetPixel(pixel_in_line, current_line,
                       Color.FromArgb(pixel_cam1, 0, pixel_cam2));
               }
              }
```

Do bitmapy je před zobrazením vytvořen zaměřovací terč pro kalibraci kamer a zároveň jsou vytvořeny modré vertikály, které reprezentují naměřené hodnoty jednotky. Pro přístup k objektům GUI z jiného vlákna se používá funkce *IVSC* (Invoke via sync context). Funkce je definována následovně:

```
void IVSC(Action uiAction)
        {
            _syncContext.Post(o =>
            {
                if (!IsDisposed) uiAction();
            }, null);
        }
```

A příklad použití je následovný:

```
IVSC(() => pictureBox_cam1.Image = cam1_bitmap);
IVSC(() => pictureBox_cam1.Invalidate());
IVSC(() => pictureBox_cam1.Update());
```

Tímto způsobem jsou aktualizovány obrázky v uživatelském rozhraní.

### 6.2.5 Driver pro LabView

Knihovna pro LabView je psaná v jazyce C# Class library. Pro použití je nutné knihovnu *LabView_driver.dll* importovat do programu. Knihovna obsahuje tyto funkce:

```
Error Connect(int port_number)
Error Disconnect(int port_number)
int Get_port_number()
void Set_cross()
void Reset_cross()
void Set_vizuals()
void Reset_vizuals()
Error Update_Content()
byte[] Get_Left_data()
byte[] Get_Right_data()
byte[] Get_Measuerd_data()
Bitmap Get_Left_image()
Bitmap Get_Right_image()
Bitmap Get_Diff_image()
Bitmap Get_Mixed_image()
int Get_Vertical(int index)
```

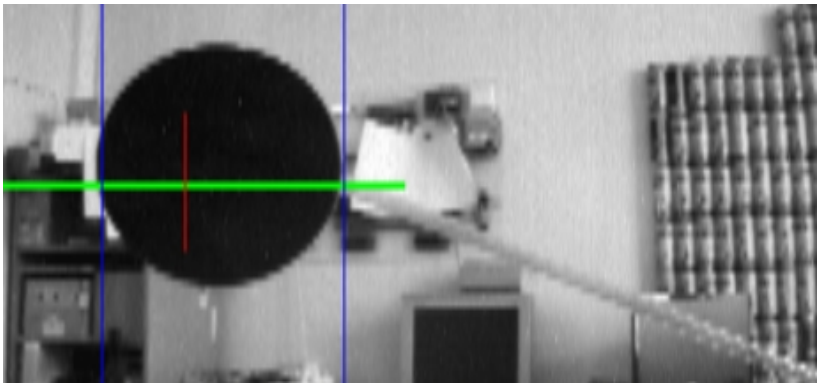
Při programovaní aplikace, je nutné nejdříve zavolat funkci *Connect*(). Tato funkce otevře virtuální sériový port. Po úspěšném připojení se zavolá funkce *Update_Content*(), která stáhne data z jednotky. Po této operaci funkce *Get_xxx*() vracejí aktuální data. Pro ukončení komunikace se volá funkce *Disconnect*(), která sériový port uvolní. Ukázka použiti knihovny je následující:

```
unsigned char *data_left;
unsigned char *data_right;
....
If ( Connect( Get_port_number( ) ) < 0 )
{
Return -1;
}
...
If ( Update_Content ( ) > 0 )
        {
        data_left = Get_Left_data ( );
        data_right = Get_Right_data ( );
}
    else    {
        return -1;
}
...
Disconnect ( Get_port_number( )  );
...
```

Ukazatel *data* ukazuje na jednobajtové pole o velikosti 752x60. V těchto datech nalezneme nasnímaný obraz z kamer.

# 7 Měření

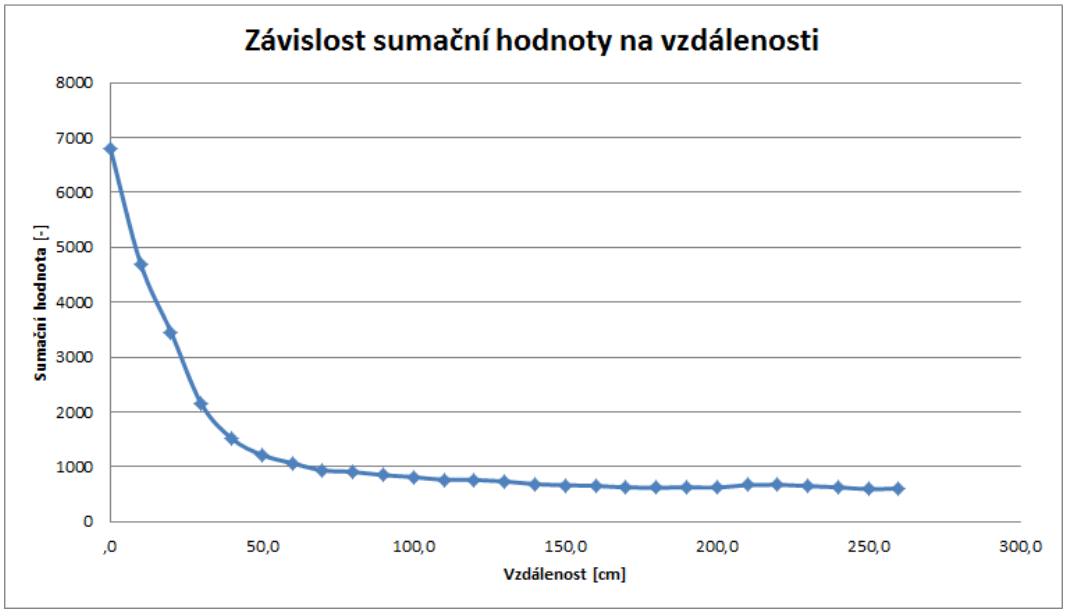
Měření jsem prováděl s využitím obou zmiňovaných metod. Jako objekt jsem použil černý kruh s průměrem 10cm (viz obr. 7.1). Postupně ze vzdálenosti tří metrů jsem objekt přibližoval k jednotce s kamerami. Mezitím jsem zapisoval naměřené hodnoty, které se zobrazují na uživatelském rozhrání programu ImageGrabber.



Obr. 7.1: Měřená překážka a její zobrazení v programi ImageGrabber

## 7.1 Měření závislosti, sumační metody na přiblížení

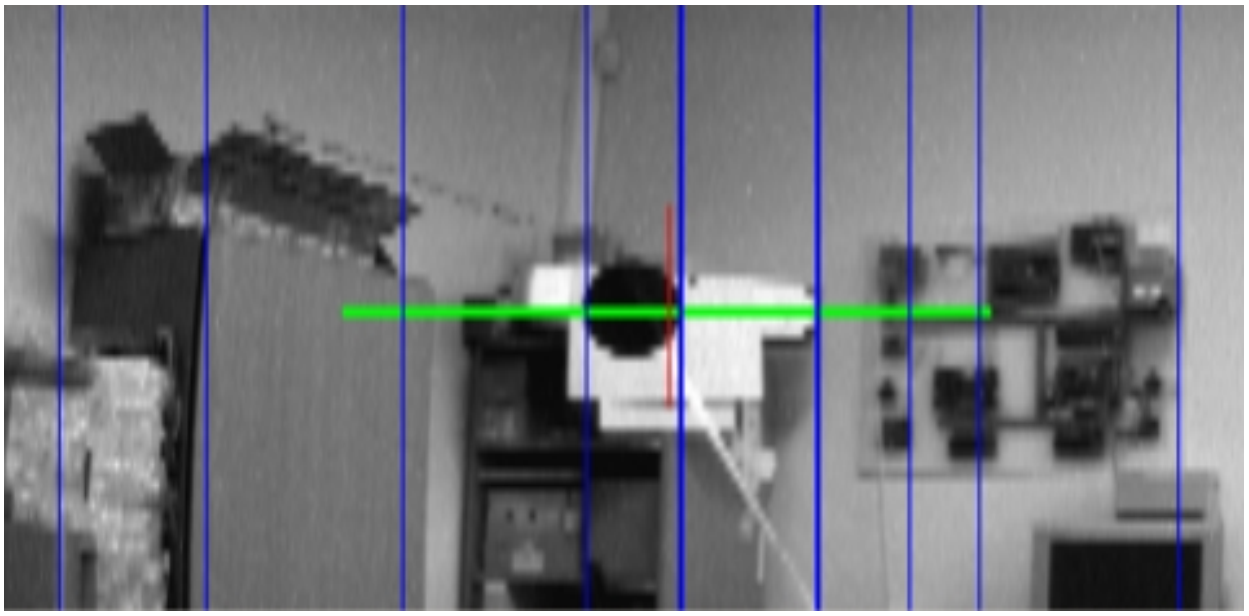
Naměřené hodnoty sumační metody nalezneme v tabulce 7.1. Závislost sumační hodnoty na vzdálenosti je zobrazena na obr. 7.2. Porovnáním se simulací nám závislost vychází zhruba stejně. Nicméně se ukázaly problémy s komplexním pozadím, kde detekce hran selhávala (viz obr. 7.4). V průbězíh měření, jsem si všímal i rozptylu hodnot, který byl silně závislý na aktuálním osvětlení místnosti. Tuto metodu lze používat pro vzdálenosti 50cm a méně. Pro větší vzdálenosti vlivem kolísaní měřené hodnoty jde stěží určit vzdálenost překážky. Další nevýhodou této metody je nepřesná lokace překážky (viz obr. 7.3).



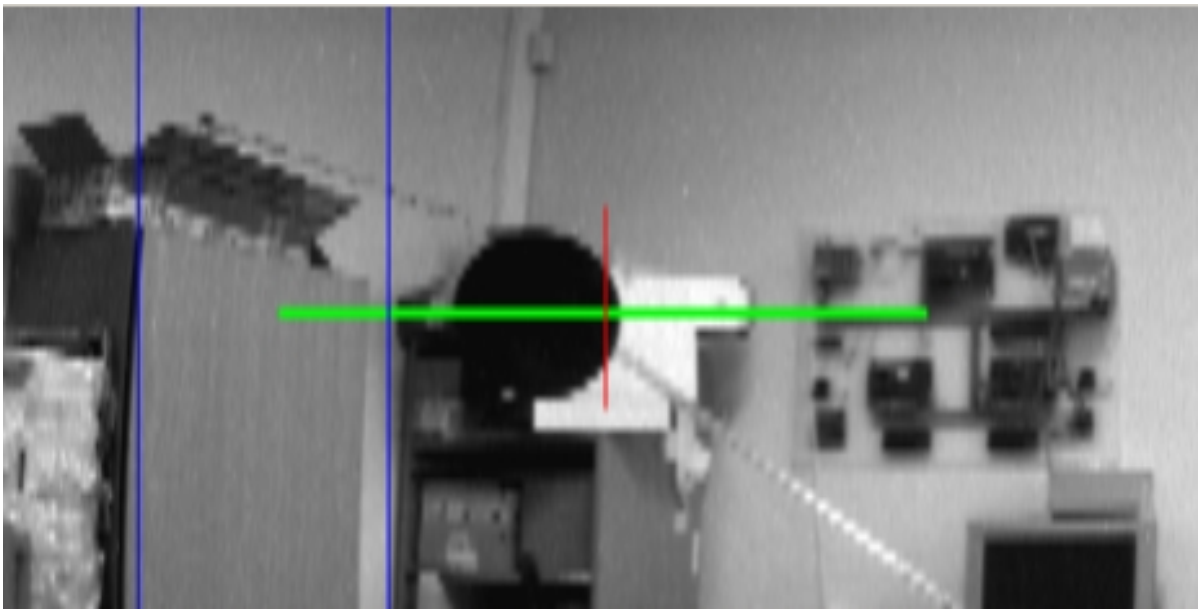

Obr. 7.2: Graf závislosti sumační hodnoty na vzdálenosti

| Vzdálenost [cm] | Hodnota sumáře [-] | Průměrný jas v prostředním řádku [-] |
|---|---|---|
| 260 | 610 | 136 |
| 250 | 594 | 133 |
| 240 | 631 | 136 |
| 230 | 651 | 138 |
| 220 | 676 | 133 |
| 210 | 670 | 131 |
| 200 | 625 | 130 |
| 190 | 630 | 130 |
| 180 | 620 | 127 |
| 170 | 627 | 124 |
| 160 | 656 | 128 |
| 150 | 668 | 128 |
| 140 | 686 | 127 |
| 130 | 737 | 134 |
| 120 | 766 | 137 |
| 110 | 765 | 130 |
| 100 | 816 | 131 |
| 90 | 852 | 128 |
| 80 | 910 | 130 |
| 70 | 938 | 124 |
| 60 | 1068 | 127 |
| 50 | 1220 | 121 |
| 40 | 1519 | 121 |
| 30 | 2151 | 121 |
| 20 | 3456 | 132 |
| 10 | 4700 | 130 |
| 0 | 6800 | 130 |

Tab. 7.1: Tabulka naměřených hodnot sumační metody



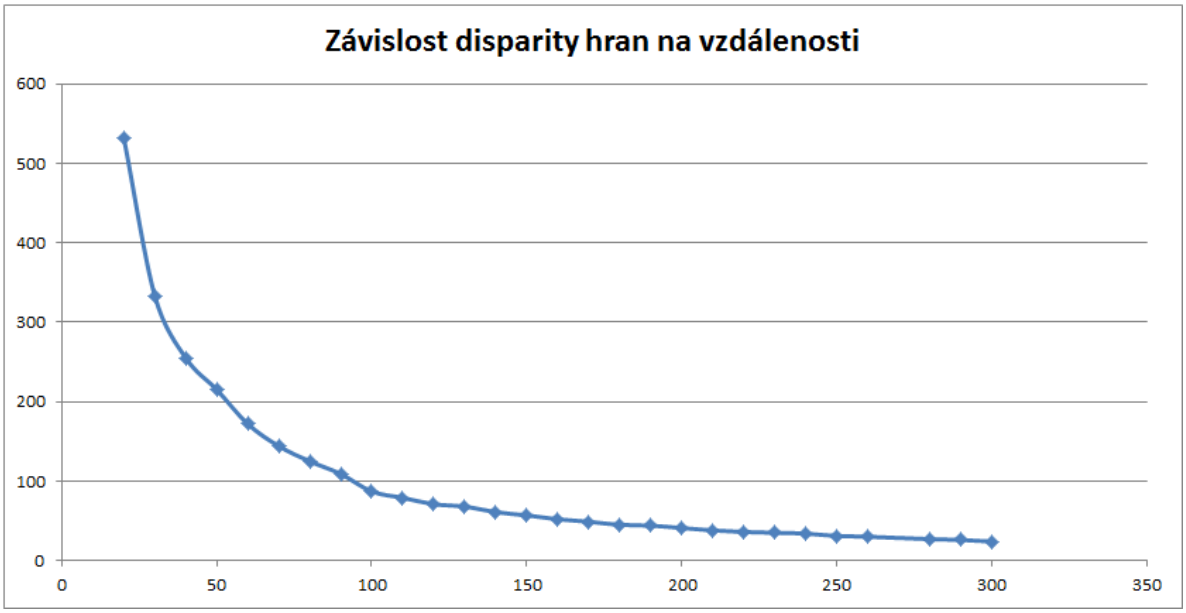
Obr. 7.3: Obraz ze sumační metody s dobře detekovanou hranou překážky



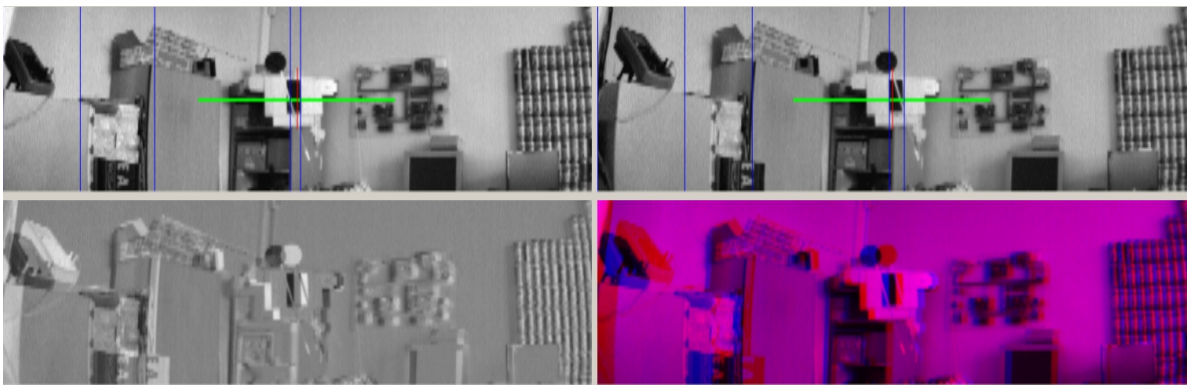
Obr. 7.4: Obraz ze sumační metody se špatně detekovanou hranou překážky

### 7.1.1 Měření vzdálenosti metodou lokální disparity obrazů

Naměřené hodnoty disparit jsou v tabulce 7.2. Závislost disparity hran na vzdálenosti je na obrázku 7.5. Tato metoda se chová mnohem spolehlivěji. Nejdříve ze všeho jsem otestoval funkčnost na testovacím obrázku ve tvaru obdélníku (viz obr 7.6). V této situaci byla průměrná disparita obdélníku 16 pixelů. Tato hodnota je patrná zejména na spodních dvou obrazcích. Vytištěný obdélník na papíru A3 byl ve vzdálenosti 340cm od kamer. Kruhový objekt byl zaměřen též, v okamžiku, kdy střední řádek jej protínal (viz obrázky meření 7.7-7.10).



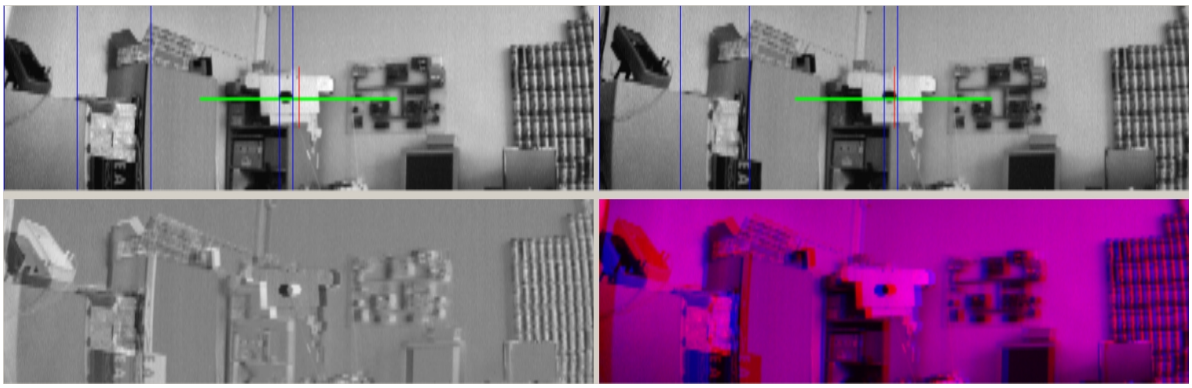

Obr. 7.5: Graf závislosti disparity hran na vzdálenosti



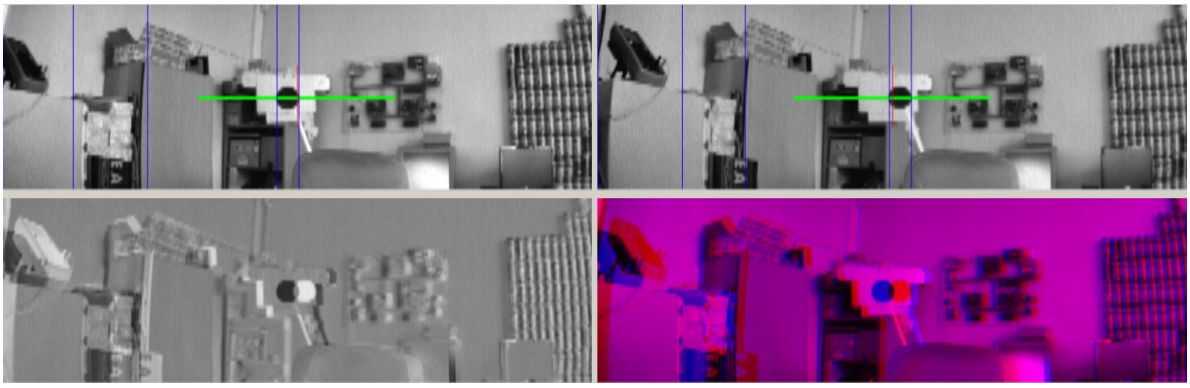
Obr. 7.6: Testovací předmět (obdelník) na vzdálenosti 340cm s metodou disparity

| Vzdálenost [cm] | Hodnota disparity [px] | Průměrný jas v prostředním řádku [-] |
|---|---|---|
| 300 | 24 | 139 |
| 290 | 26 | 139 |
| 280 | 27 | 139 |
| 260 | 30 | 134 |
| 250 | 31 | 133 |
| 240 | 34 | 131 |
| 230 | 35 | 130 |
| 220 | 36 | 130 |
| 210 | 38 | 129 |
| 200 | 41 | 132 |
| 190 | 44 | 131 |
| 180 | 45 | 128 |
| 170 | 49 | 129 |
| 160 | 52 | 129 |
| 150 | 57 | 128 |
| 140 | 61 | 128 |
| 130 | 68 | 126 |
| 120 | 71 | 126 |
| 110 | 79 | 130 |
| 100 | 87 | 128 |
| 90 | 109 | 135 |
| 80 | 125 | 139 |
| 70 | 144 | 137 |
| 60 | 172 | 140 |
| 50 | 215 | 129 |
| 40 | 255 | 136 |
| 30 | 333 | 134 |
| 20 | 532 | 136 |

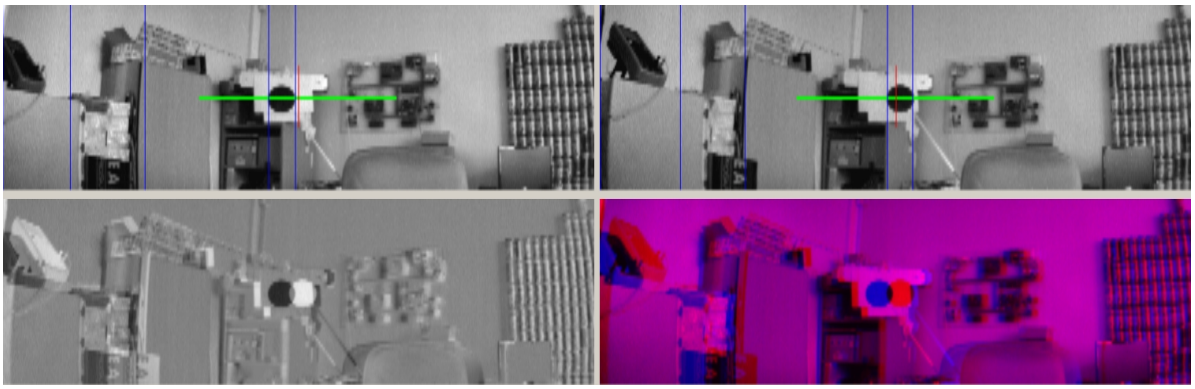
Tab. 7.2: Tabulka naměřených hodnot metody disparit



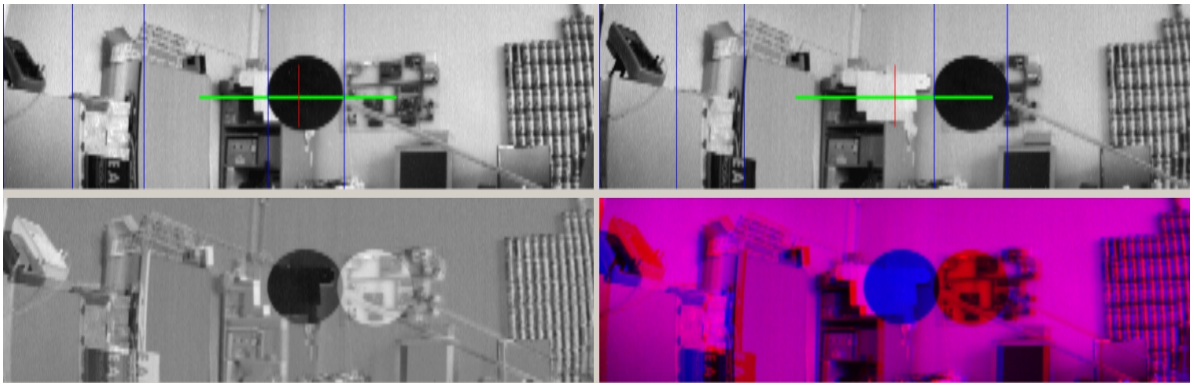
Obr. 7.7: Ukázka 1 detekce předmětu metodou disparity.



Obr. 7.8: Ukázka 2 detekce předmětu metodou disparity.



Obr. 7.9: Ukázka 3 detekce předmětu metodou disparity.



Obr. 7.10: Ukázka 4 detekce předmětu metodou disparity.

# 8 Závěr

Úspěšně jsem zhotovil a oživil hardware stereo snímače pro vyhodnocení překážek. Podle měření je metoda disparity mnohem přesnější a lepší. Dokáže nejen určit vzdálenost překážky, ale i její horizontální polohu. Výhodou metody je, že neklade vysokou náročnost a vykazuje dostatečnou spolehlivost pro praktické využití. Metoda není nijak citlivá na intenzitu osvětlení jak to bylo u metody sumace. Nevýhodou jsou překážky bez hran s gradientním nebo homogenním povrchem. Další problém tvoří povrch s periodicky se opakující kontrastní texturou.

Elektronika jednotky je vyrobena z běžně dostupných součástek a je navržena tak, aby se dala opakovaně vyrábět a použít i pro jiné účely. Může sloužit jako elektronická pomůcka při vývoji a studii. Jednotka je poskládána z modulu, které lze snadno demontovat a nahradit jinými. Jednotka může být napájena z USB portu. Pro práci nevyžaduje externí napájení ani v plném provozu při nahrávaní programu do procesoru.

Programové vybavení jednotky je na vysoké úrovni a využívá běžně dostupných prostředků. Kombinace programovacího jazyka C pro procesor s kombinací jazyka C# pro počítač, vytváří silný nástroj pro jednoduchou a rychlou práci. Knihovna pro virtuální systémy LabView otvírá další možnosti v oblasti průmyslového využití.

V porovnání s Kinetic je tato jednotka pouhým jednoduchým nástrojem, nicméně postačující pro použití v oblasti robotiky a stereoskopie jako levná a jednoduchá alternativa. Kinetic ovšem není stavěn pro průmyslové použití.

Na této práci jsem se naučil jak postupovat v případě složitých projektů. Naučil jsem se pracovat s novou řadou procesoru STM32F2 a s obrazovým senzorem.

V prací jsem nestihl dořešit časovou synchronizaci pomocí protokolu PTP. Důvodem je nutnost hardwarové úpravy a výroby nového plošného spoje. Dále jsem nestihl zápis snímků na paměťové médium. Knihovny pro tuto funkci jsou napsány, ale hardware není vybaven slotem pro SD kartu.

Diky tomu, že se v rámci tohoto projektu podařilo úspěšně zvládnout vývoj aplikace tohoto nového procesoru i jeho spolupráci s obrazovým senzorem CMOS, může tato diplomová práce a veškerá v ní obsažená dokumentace sloužit také jako vzorové řešení pro další bakalářské a diplomové práce, které se budou věnovat vestavěným aplikacím obrazových senzorů pro snímání polohy objektů.

Tato práce byla jedna z prvních aplikací na procesoru STM32F207 v České republice mimo vývojové pracoviště firmy ST. Projekt využíval vzorek prototypové série s chybami. Podpora pro tuto řadu procesoru byla problematická. Bylo nutné vyhledat správné vývojové prostředí, které bylo nutné opravit. Výroba procesoru byla ve skluzu a v případě fatální chyby by obstarání nového kusu bylo problematické. První vzorek procesoru jsem osazoval ve druhém týdnu semestru.

Případná další práce na projektu je určitě možná, zejména v algoritmech pro disparitu obrazů. Lze také použít známý algoritmus autokorelace, který by mohl být také užitečný v tomto případě. V procesoru je dostatek místa pro další řízení a obsluhu. Lze doprogramovat regulátor pro pohon robota a vytvořit tak funkční koncept. Návrh jednotky umožňuje přidání dalších součástek a tak lze projekt patřičně rozšiřovat.

# 9 Literatura

[1] *Chloupkova, Tereza:* **Fyziologické principy procesu vidění – tvorba a vnímání obrazu** MASARYKOVA UNIVERZITA, Přírodovědecká fakulta 2007

[2] *Microsoft Kinect, Wikipedie* http://en.wikipedia.org/wiki/Kinect

[3] *Gimp, Konvoluční matice* http://docs.gimp.org/2.2/cs/plug-in-convmatrix.html

[4] *Hledání hran, Václav Hlaváč* http://cmp.felk.cvut.cz/ hlavac/TeachPresCz/11DigZprObr/22Detek

[5] *Kalová, Ilona:* **Segmentace a detekce geometrických primitiv – Počítačové vidění** Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií

[6] *CCD snímač, Wikipedie* http://cs.wikipedia.org/wiki/CCD

[7] *Fischer, Jan:* **Obrazové senzory CCD, CMOS – katedra měření** České vysoké učení technické, Fakulta elektrotechnická 2009

[8] *Dokumentace, STM32F207-107 a STM3210C-EVAL* http://www.st.com

[9] *Dokumentace, MT9V032 a MT9M002* http://www.micron.com

[10] *National Semiconductor, DS92LV16 Design Guide* http://www.national.com 2002

[11] *Matsumoto, Yoshio IEEE:00619071* **Real-time Color Stereo Vision System for a Mobile Robot based on Field Multiplexing** Inoue-Inaba Lab., Dept. of Mechano-Informatics, Faculty of Engineering 1997

[12] *Massimo, Bertozzi IEEE:00650851* **GOLD: A Parallel Real-Time Stereo Vision System for Generic Obstacle and Lane Detection** IEEE Transactions on image processing 1998 January

# A Seznam použitých zkratek

**DMA** Direrct memory access

**SDIO** Secure digital input output

**MMC** Multimedia card

**SD** Secure digital

**USB** Universal serial bus

**PC** Personal computer

**IRQ** Interrupt request

**DSP** Digital signal processing

**CCD** Charge-coupled device

**CMOS** Complementary metal–oxide–semiconductor

**I2C** Inter-Integrated Circuit

**HSYNC** Horizontal synchronization

**VSYNC** Vertical synchronization

**PIXCLK** Pixel clock

**CPU** Central processing unit

**SCLK** System clock

**GDB** GNU project debugger

**GNU** GNU's Not Unix!

**JTAG** Joint test action group

**SWD** Serial wire debug

**DP** Diplomová práce

**RTC** Real time clock

**LED** Light emiting diode

**SMD** Surface mount device

**RAM** Random access memory

**CAM_OE** Camera output enable

**AIN** Analog in

**PTP** Precision Time Protocol

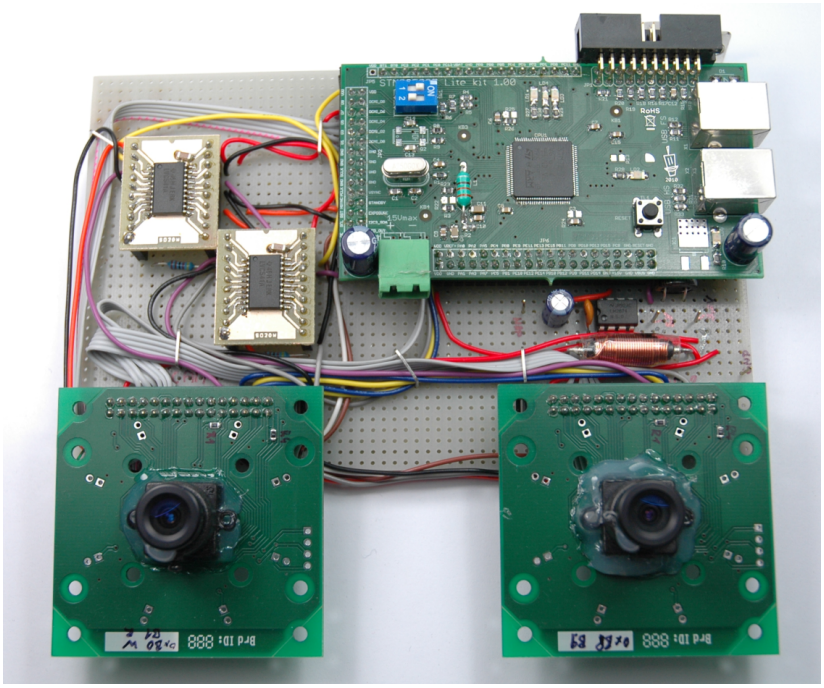
**IIR** Infinite impulse response

**PTP** Precision Time Protocol

**PTP** Precision Time Protocol

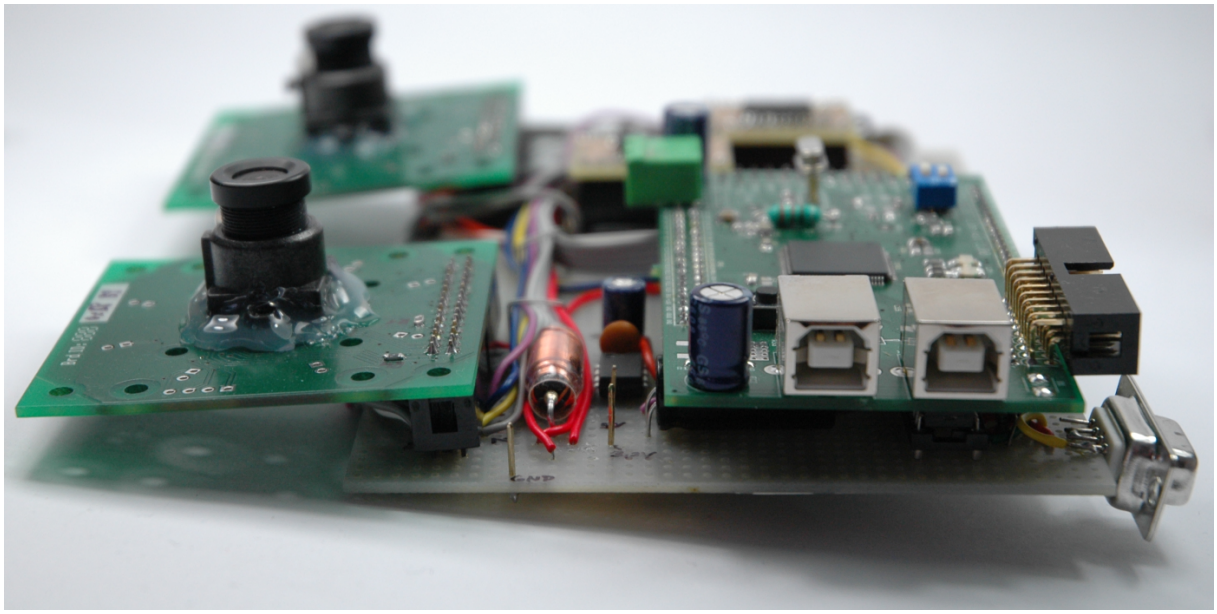
⋮

# B Ukázka hotové jednotky stereo snímače

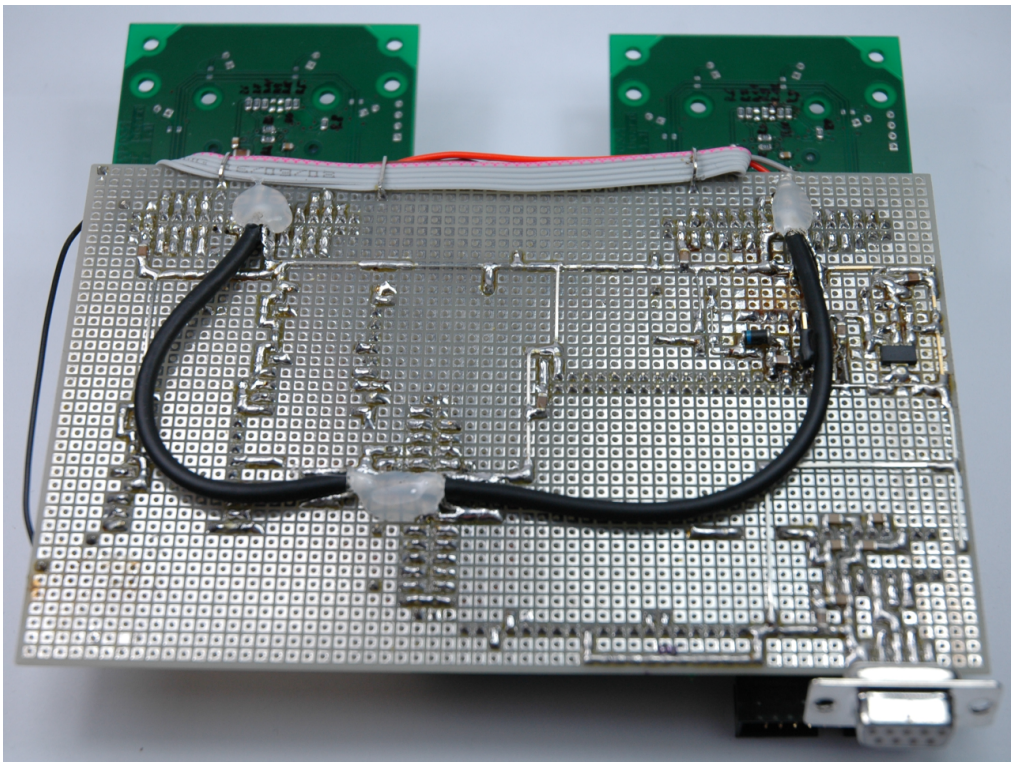


Obr. B.1: Jednotka stereo snímače, pohled zepředu

Obr. B.2: Jednotka stereo snímače, pohled na kameru snímače CMOS



Obr. B.3: Jednotka stereo snímače, pohled z boku



Obr. B.4: Jednotka stereo snímače, pohled na zadní stranu

# C Obsah přiloženého CD

Na výpisu níže je zobrazena stromová struktura adresářů na CD.

```
cd
|-- Foto
    |-- fotky_jednotky
    |-- metoda 1
    |-- metoda 2
|-- Hardware
    |-- BaseBoard
    |-- STM32F207 (Lite kit)
         |-- Firmware
             |-- F217test
                 |-- CameraTest
|-- Keil
    |-- ST-Link
|-- LaTeX
    |-- graphics
|-- Literatura
|-- Matlab
    |-- Analyza
|-- Software
    |-- ImageGrabber
    |-- LabView_driver
```

Adresář `Foto` obsahuje screenshoty a fotky jednotky z měření.

Adresář `Keil` obsahuje driver pro ST-Link k funkčnosti KEIL verze 4.20 s procesorem STM32F207.

Obsahem adresáře `LaTex` jsou kompletní zdrojové kódy tohoto dokumentu vytvořené typografickým systémem LaTeX. V `graphics` se nachází všechny obrázkové přílohy.

V adresáři `Matlab` nalezneme veškeré simulované data.

Datasheety obvodů použitých u projektu najdeme v adresáři `Literatura`. Schémata ve PDF formáte jsou k dispozici v adresáři `Hardware`

Adresář `Literatura` obsahuje také pramen informacií ze kterého byla tato prace sestavěna.