VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INTELLIGENT SYSTEMS

# SIMULÁTOR STŘELNICE S VYUŽITÍM ZPRACOVÁNÍ OBRAZU

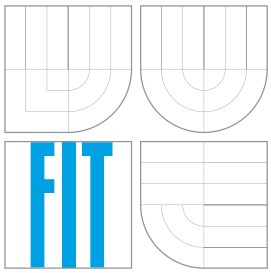
BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

PAVEL CSÓKA

BRNO 2013

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INTELLIGENT SYSTEMS

# SIMULÁTOR STŘELNICE S VYUŽITÍM ZPRACOVÁNÍ OBRAZU

SHOOTING RANGE SIMULATOR WITH IMAGE PROCESSING

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — PAVEL CSÓKA
AUTHOR

VEDOUCÍ PRÁCE — Ing. ALEŠ MARVAN
SUPERVISOR

BRNO 2013

# Abstrakt

Tato práce se zabývá návrhem a praktickou realizací simulátoru střelnice s využitím zpracování obrazu. Střelnice je postavena na principu vysvěcování laseru umístěného ve zbrani na zobrazovací plochu terče (dataprojektor nebo LCD) a jeho detekci v obraze získaném kamerou, která tuto plochu snímá. Pro detekci laseru je navrhnut a implementován software v jazyce C++. Využívá se přitom OpenCV knihovna počítačového viděni. Dále je popsána úprava airsoftové zbraně a návrh, vytvoření a umístění řídící jednotky pro její ovládání. K tomuto je využit mikrokontrolér STM32, pro který je navrhnut a implementován řídící software v jazyce C. Jsou zde popsány použité techniky zpracování obrazu a funkce mikrokontroléru. Pro vyhodnocení funkčnosti řešení jsou provedeny praktické testy. V závěru je uvedeno zhodnocení práce a možnosti vylepšení a rozšíření vytvořeného systému.



# Abstract

This thesis is aiming on design and implementation of Shooting range simulator with image processing. It is based on detection of a laser spot on the projection plane where the target is displayed (dataprojector or LCD) by using camera which captures image of this plane. It designs and implements the detection software using C++ language and OpenCV computer vision library and describes gun modification, which consists of design and creation of control unit for this gun and fitting it in. Control unit is based on STM32 microcontroller device, its firmware implementation is done using C language. It describes approaches in computer vision applications used in this thesis and main features of used microcontroller. Practical test are made to analyze system's function. In the final chapter is the summarization of this thesis and suggestions to further development and improvement of this system.



# Klíčová slova

STM32, OpenCV, simulátor střelnice, laserová střelnice, zpracování obrazu, úprava airsoftové zbraně

# Keywords

STM32, OpenCV, shooting range simulator, laser shooting range, image processing, airsoft gun modification

# Citace

Pavel Csóka: Simulátor střelnice s využitím zpracování obrazu, bakalářská práce, Brno, FIT VUT v Brně, 2013

# Simulátor střelnice s využitím zpracování obrazu

## Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Aleše Marvana

.....
Pavel Csóka
14. května 2013

## Poděkování

Děkuji vedoucímu práce, panu Ing. Aleši Marvanovi, za ochotu, poskytnutí informačních zdrojů a odbornou pomoc. Poděkování patří také mým rodičům za podporu při studiu a tvorbě této práce.

# Obsah

# Kapitola 1

# Úvod

Účelem této práce je vytvoření simulátoru střelnice, který je možné použit jako trenažer střelby ze zbraně sloužící jako levnější varianta k tréninku střelby z palných zbraní.

Záliba ve střelných zbraních byla vždy velmi vysoká, ale ne každý má možnost se dostat na střelnici, aby si jí mohl vyzkoušet. Může to být ať už z důvodů zdravotních nebo jen nedostupností těchto střelnic, protože mají specifické požadavky na prostory, kde se mohou nacházet. Nejdůležitějším faktorem je bezpečnost, která musí být zajištěna při použití střelných zbraní. Tyto požadavky na prostory a bezpečnost jsou u tohoto simulátoru mnohem menší a jeho provozování může být uskutečněno prakticky kdekoliv, kde je potřebné místo. To znamená, že se může provozovat i v budovách v zástavbě, kde by bylo použití klasické střelnice nemyslitelné.

Simulátor střelnice je založen na principu rozpoznávání obrazu, ve kterém se za pomocí softwaru v počítači vyhledává stopa laseru na ploše, kde je zobrazen terč. K tomu se využívá kamera snímající plochu terče a samotné vysvěcování laseru simuluje střelbu zbraně. Proto se takovéto střelnice označují jako „laserové střelnice“. Ke střelbě se využívá upravená airsoftová zbraň osazená laserem a ovládaná mikrokontrolérem.

Úvodní dvě kapitoly popisují použitý mikrokontrolér (kapitola 3) a knihovnu použitou ke zpracování obrazu (kapitola 2). V podkapitole 2.3 jsou popsány základní techniky využívané v rozpoznávání obrazu a uplatněné při tvorbě simulátoru střelnice.

V této práci jsem navrhl softwarovou aplikaci pro rozpoznávání zásahů laseru (jeho vysvícení) do plochy terče a zařízení pro ovládání airsoftové zbraně. Tyto návrhy jsou popsány v kapitole 4. Následně jsem navržené softwarové řešení implementoval, což je popsáno v kapitole 5.

Stavba ovládacího zařízení zbraně a jeho vestavění do zbraně a další potřebné úpravy na zbraní jsou popsány kapitolou 6. Zároveň je zde popsána implementace softwaru mikrokontroléru (*firmwaru*), kterým je realizována jeho požadovaná funkčnost.

V kapitole 7 jsou popsány provedené praktické testy tohoto navrženého, implementovaného a vytvořeného simulátoru střelnice.

V poslední kapitole 8 je napsán závěr práce, který obsahuje zhodnocení vytvořeného systému a možnosti jeho dalšího rozšíření a vylepšení.

# Kapitola 2

# Popis knihovny OpenCV a technik zpracování obrazu

OpenCV (Open Source Computer Vision Library)[1] je open source knihovna (soubor programových funkcí) pro tvorbu softwaru počítačového vidění a strojového učení. Byla vytvořena za účelem poskytnutí obecné základny pro aplikace využívající počítačového vidění a urychlila tím jejich vývoj a nasazení. Je vydána pod BSD[1] licencí, která dovoluje provádět úpravy v kódu a umožňuje její použití jak v komerčních, tak nekomerčních aplikacích.

Knihovna obsahuje více jak 2500 optimalizovaných klasických i současných nejmodernějších algoritmů pro aplikaci počítačového vidění a strojového učení. Je vysoce využívána po celém světě a stále v aktivním vývoji. Může být použita např. pro detekci a rozpoznávání obličejů, objektů, sledování pohybujících se objektů, pro rozšířenou realitu (*augmented reality*), v robotice, atd.

Obsahuje rozhraní (API[2]) pro programovací jazyky C, C++ a Python a je podporována na operačních systémech Windows, Linux, Android a Mac OS X. OpenCV je implementována v jazyce C++ a bezproblémově funguje s STL kontejnery. Její největší uplatnění je v real-time aplikacích zpracovávajících obraz díky tomu, že využívá nejmodernějších technologií urychlujících samotné výpočetní úkony – MMX, SSE[3], paralelní zpracovávání na víceprocesorových systémech, OpenCL[4] pro výpočty na grafické kartě a další.

Největší výhodou je velmi jednoduchá aplikace algoritmů a jejich efektivní zpracovávání, díky vysoké optimalizaci.

## 2.1 Struktura knihovny

OpenCV má modulární strukturu, což znamená, že se skládá z několika sdílených nebo statických knihoven. OpenCV obsahuje následující moduly[2]:

- **core** – kompaktní modul definující základní datové struktury a funkce využívané všemi ostatními moduly.
- **imgproc** – modul pro zpracování obrazu, který implementuje lineární a nelineární funkce pro filtraci obrazu, geometrické transformace, konverzi barevného rozsahu, histogramy a další.

[1] Popis licence k nalezení na http://www.gnu.org/licenses/license-list.html#ModifiedBSD
[2] API je aplikační programové rozhraní
[3] MMX, SSE jsou rozšíření instrukční sady procesorů
[4] OpenCL je jazyk pro paralelní výpočty

- **video** – modul pro analýzu videa, který obsahuje algoritmy pro vyhodnocování pohybu, sledování objektů a pro extrakci pozadí.
- **calib3d** – základní geometrické algoritmy pro vícenásobné pohledy, kalibraci jedné nebo více kamer, vyhodnocování pozice objektu a elementy pro 3D rekonstrukci scény.
- **features2d** – detekce významných vlastností scény, jejich popis a zařazení.
- **objdetect** – detekce objektů a předdefinovaných konkrétních prvků (například obličeje, oči, lidé, auta).
- **highgui** – jednoduché rozhraní pro zachytávání videa, obrazů a video kodeků.
- **gpu** – algoritmy zpracovávané přes grafickou kartu z různých jiných OpenCV modulů.
- **a další**

## 2.2 Aplikační rozhraní API

API využívá jmenný prostor (*namespace*) `cv::`, ve kterém jsou vloženy všechny třídy a funkce. Má automatickou správu paměti založenou na počítadle odkazů na danou část paměti. Jedině pokud je počítadlo na nule je paměť, která již není nikým využívána, uvolněna. Implementuje tzv. Saturation aritmetiku, která zajišťuje správnou úpravu hodnoty do daného rozsahu proměnné, pokud je hodnota mimo rozsah této proměnné. Jestli je tato hodnota menší než spodní hranice povoleného rozsahu, tak se zapíše nejnižší hodnota z tohoto rozsahu a obráceně.

Pro vypořádávání s chybami používá systém výjimek (*Exceptions*). Současná implementace OpenCV umožňuje aby aplikace pracovala s více vlákny[2].

## 2.3 Popis technik zpracovávání obrazu

V této sekci popíši některé techniky využívané v počítačovém vidění, které najdou uplatnění při řešení této práce.

### 2.3.1 Transformace obrazu

Transformací obrazu se rozumí jeho přeměna z jedné formy reprezentace do jiné. V této sekci bude popsáno prahování obrazu a převod barevného obrazu do černobílého, tj. obrazu intenzit.

#### Prahování obrazu

Prahování (*thresholding*) je operace, při které se ze vstupního jednokanálového obrazu ($src(x, y)$), tj. obrazu intenzit, získá obraz binární ($dst(x, y)$). Tento se získá aplikací prahovací funkce (2.1) na každý pixel vstupního obrazu podle zvolené hodnoty mezní intenzity, neboli prahu. Prahování je také možno využít pro odstranění nežádoucího šumu v obraze, tj. odstranění pixelů s příliš velkou (2.2) nebo malou intenzitou (2.3). Tuto operaci zajišťuje OpenCV funkce `cv::threshold()`[2][3].

$$dst(x,y) = \begin{cases} \max & \text{pro } src(x,y) > 0 \text{ práh} \\ 0 & \text{pro ostatní hodnoty} \end{cases} \tag{2.1}$$

$$dst(x,y) = \begin{cases} \text{práh} & \text{pro } src(x,y) > 0 \text{ práh} \\ src(x,y) & \text{pro ostatní hodnoty} \end{cases} \tag{2.2}$$

$$dst(x,y) = \begin{cases} src(x,y) & \text{pro } src(x,y) > 0 \text{ práh} \\ 0 & \text{pro ostatní hodnoty} \end{cases} \tag{2.3}$$

#### Konverze barevného obrazu RGB do černobílého

Jelikož mnoho algoritmů vyžaduje na svém vstupu černobílý obraz, neboli obraz intenzit jednotlivých pixelů, musíme se vypořádat s problémem jak tento obraz vytvořit z barevného RGB obrazu. Obraz reprezentovaný v barevném prostoru RGB se skládá ze tří hlavních barevných složek (kanálů). Jak už název napovídá, jedná se o červenou (R – red), zelenou (G – green) a modrou (B – blue) složku. Intenzity těchto tří barev definují barvu výslednou a musí být uvedeny pro každý jednotlivý pixel.

Při převodu tohoto tří-složkového (kanálového) obrazu na obraz jednosložkový je část informací ztracena. Využívá se přitom vzorce (2.4):

$$Y = 0.299 * R + 0.587 * G + 0.114 * B \tag{2.4}$$

kde $Y$ je výsledná intenzita[2]. Tuto konverzi implementuje OpenCV funkce `cvtColor()`.

### 2.3.2 Detekce bodů zájmu

V počítačovém vidění je koncept vyhledávání význačných bodů (*interest points*, též *key-points* nebo *feature points*), neboli bodů, které nesou důležité a zajímavé obrazové informace, vysoce využíván. Slouží jako základ pro rozpoznávání objektů, sledování pohybů v obraze a pro další účely. Je založen na principu zvolení určitých bodů v obraze a jejich lokální analýze namísto zpracovávání obrazu jako celku. Tento přístup funguje spolehlivě jenom pokud je takovýchto bodů zájmu zvoleno dostatek a podle jejich vlastností lze jednoznačně určit co tyto body představují. Jeho výhodou je menší výpočetní náročnost. Tato sekce představí některé základní detektory bodů zájmů[3].

#### Detektor hran

Detekování hran (*Edge detection*) v obraze je jednou ze základních a nejdůležitějších částí v oboru počítačového vidění. Je to proto, že slouží jako základ pro algoritmy vyšší úrovně zpracovávání, které závisí právě na jejich dobré detekci. Obrazy obsahují mnoho informací, kde je velká část z nich nepodstatná, např. pozadí. Takže se v počáteční fázi snažíme počet těchto informací snížit a vyjmout pouze pro nás zajímavé části obrazu. Proto hledáme právě hrany, které z hlediska významu obvykle nesou cenné informace. Seskupením nalezených hran do větších celků získáme různé křivky čáry a obrysy. Toho se dá využít například k detekci objektů, jelikož jsou většinou ohraničeny viditelnými obrysy (viz kap. 2.3.3)[4][5].

Obecně se hrany nachází na hranicích mezi oblastmi rozdílných barev, intenzity nebo textury.[4] V počítačovém vidění se využívá rozdílu intenzit, kdy nejprve převedeme barevný obraz na černobílý, který určuje intenzitu černé a bíle barvy. Jelikož rozdělení obrazu na takovéto oblasti je složitý úkon, využívá se pouze lokálních informací. To znamená, že hranu definujeme jako oblast prudké změny intenzity. V reálných podmínkách tato změna

neprobíhá skokově, ale postupně. Matematicky tuto změnu a její směr vyjádříme pomocí gradientu (vzorec (2.5)).

$$J(x) = \left(\frac{\partial I}{\partial x}, \frac{\partial I}{\partial y}\right)(x) \tag{2.5}$$

Vektor gradientu $J$ vždy ukazuje ve směru nejprudší změny intenzity a jeho velikost vyjadřuje jak velká tato změna je. V obraze to znamená, že jeho směr je kolmý k hraně a směřuje od menší intenzity po větší (od tmavší po světlejší). Jelikož derivací obrazu se zvýší nežádoucí šum, je ho potřeba odstranit ještě před výpočtem gradientu. K tomuto se používá Gaussův filtr[4].

### Canny detektor hran

Jedním z algoritmů pro detekci hran, který je implementován v OpenCV je Canny86[6] Je založen na Sobel operátoru, což je lineární filtr (filtr je lineární, když hodnota nahrazovaného pixelu je rovna váženému součtu okolních pixelů) pro detekci hran. Má strukturu 3 x 3 matice, která udává násobící koeficient pro sousední pixely a střed matice udává pixel, na který je filtr aplikován. Tato matice se nazývá **kernel** a její aplikace na každý pixel obrazu se nazývá **konvoluce**.

$$\begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix} \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix} \tag{2.6}$$

Operátor (2.6) se aplikuje na obraz ve dvou směrech – vertikálním a horizontálním. Jeho aplikací získáme míru změn intenzit v těchto směrech a tudíž oblasti výskytu hran. Matematicky je vyjádřen pomocí vektoru gradientu (2.5)

Po provedení konvoluce obrazu Sobelovým kernelem získáme 2D vektor gradientu. Vypočteme jeho normu, která udává velikost změn intenzity. Tato se počítá jako Eukleidovská norma (též zvaná L2 norma):

$$|J(x)| = \sqrt{\left(\frac{\partial I}{\partial x}\right)^2 + \left(\frac{\partial I}{\partial y}\right)^2} \tag{2.7}$$

V počítačovém vidění se tato norma počítá jako součet absolutních hodnot změn intenzit. Nazývá se L1 norma (2.8) a blíží se přesnosti normy L2 pří nižší náročnosti na výpočet.

$$L1 = abs(sobelX) + abs(sobelY) \tag{2.8}$$

Výsledný binární obraz (mapu hran) získáme prahováním normovaného obrazu. Musí se zvolit správná velikost prahu, abychom získali dobrý výstup. Pokud je práh příliš nízký, tak je zachováno příliš mnoho (hrubých) hran a naopak, pokud je moc vysoký budou hrany roztříštěné.

Problém volby správné hodnoty prahu řeší Canny operátor, který představuje koncept tzv. **hysterezního prahování**. Spočívá v tom, že se k získání binární mapy použije dvou výsledků prahování při různých hodnotách prahu. Prvně se použije nižší hodnota prahu, ze které získáme velmi mnoho hran, včetně velké části nepodstatných. Poté se prahuje vyšší hodnotou, díky které dostaneme binární mapu výrazných, tzn. podstatných hran v obraze, ale zároveň některé významné mohou být ztraceny. Kombinací těchto dvou map získáme

výsledek (obr. 2.1). Algoritmus Canny tyto mapy spojí tak, že zachová pouze ty části získané malým prahem, které jsou vzájemně nepřerušeně propojené a napojí je na výsledek aplikace vyššího prahu. Tzn., že všechny části mapy získané velkým prahem jsou zachovány, zatímco izolované části z prahu malého jsou odstraněny[3].


Obrázek 2.1: Ukázka aplikace algoritmu Canny. Vlevo původní obraz, vpravo výsledná binární mapa (v tomto obrázku jsou invertovány barvy pro lepší čitelnost)[3]

Canny algoritmus je v OpenCV implementován funkcí `cv::Canny()`, která po zadáni vstupního obrazu a hodnot malého a velkého prahu vypočítá výsledný binární obraz.

### Shi-Tomasi feature detektor

Jelikož mnohdy potřebujeme z obrazu zjistit více detailů, než jenom polohu a rozmístění hran v obraze, přichází na řadu pokročilejší algoritmy pro detekci. Jedním takovým je i Shi-Tomasi feature detector[7]. Tento algoritmus se zaměřuje na vyhledávání nejvýraznějších rohů v obraze. Rohy v obraze jsou zajímavé z toho důvodu, že je jednoduché určit jejich přesnou pozici. Je to dáno tím, že jsou určeny dvěma rozměry, tj spojnicí dvou hran. Takovéto rohy lze najít např. budovách, dveřích a jiných předmětech ostrými hranami. Povětšinou tyto vlastnosti mívají objekty vytvořené člověkem a jsou tudíž obvyklými vyhledávanými body zájmů.

Shi-Tomasi detektor počítá průměrnou změnu intenzity ve všech směrech do určité vzdálenosti, určené malým oknem, kolem jednoho aktuálního bodu. Vektorem posuvu $(u, v)$ se tato změna určena vzorcem 2.9

$$R = \sum \left(I(x+u, y+v) - I(x,y)\right)^2 \tag{2.9}$$

Tento výpočet je proveden pro celé definované okolí bodu. Z toho vyplývá, že roh je definován jako bod, pro který je průměrná změna intenzity větší ve více než jednom směru. Postupuje přitom tak, že nejdříve najde směr největší změny a potom, pokud je k němu v kolmém směru změna intenzity také vysoká, je nalezen roh. Matematicky toto lze vyjádřit maticí 2.10.

$$R \approx \begin{bmatrix} u & v \end{bmatrix} \begin{bmatrix} \sum \left(\frac{\partial I}{\partial x}\right)^2 & \sum \left(\frac{\partial I}{\partial x}, \frac{\partial I}{\partial y}\right)(x) \\ \sum \left(\frac{\partial I}{\partial x}, \frac{\partial I}{\partial y}\right)(x) & \sum \left(\frac{\partial I}{\partial y}\right)^2 \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} \tag{2.10}$$

Tato matice se nazývá **kovarianční matice** a udává míru změn intenzit ve všech směrech. Vyjádřením vlastních čísel matice získáme maximální průměrnou změnu intenzity a k ní hodnotu v kolmém směru. Pokud jsou tyto dvě hodnoty vysoké, nacházíme se na pozici rohu. Podmínka přijetí takového bodu jako rohu závisí na velikosti těchto hodnot, kdy nižší z nich musí mít vyšší hodnotu než je stanovená hodnota prahu. Obdobně, pokud jsou obě hodnoty nízké, tak se nacházíme v relativně stejnorodém prostoru a pokud je jedna z nich vyšší, tak se nacházíme na pozici hrany.

Shi-Tomasi počítá vlastní hodnoty matice přímo, jelikož hlediska výkonu na dnešních procesorech je tento výpočet již zanedbatelný. Jako další faktory zlepšující detekci bodů zájmu zavádí možnost omezení maximálního počtu nalezených bodů, kdy jsou vybrány pouze ty nejkvalitnější, a minimální vzdálenost mezi těmito body[3].

Shi-Tomasi je v Opencv implementován pomocí funkce `goodFeaturesToTrack()`.

### 2.3.3 Strukturální analýza

Strukturální analýza se zabývá vyhledáním jednotlivých logických celků v obraze na základě kterých je obraz rozčleněn a také vlastností těchto celků. Bude zde popsáno vyhledání obrysů (kontur) a informací, které z těchto celků lze získat.

#### Vyhledávání kontur v obraze

Seskupením nalezených hran v obraze do propojených celků získáme ohraničení (obrysy, kontury) jednotlivých objektů. Díky tomu jsme schopni rozdělit obraz do logických celků, které jsou určeny právě těmito konturami. Nalezené hrany jsou uloženy ve formě binárního obrazu, ze kterého se vychází. Nalezení těchto kontur je provedeno pomocí algoritmu Suzuki85[8].Tento algoritmus hledá vzájemně propojené složky obrazu (*connected components*) a vytváří mezi nimi hierarchickou strukturu. V binárním obraze to jsou sousedící pixely, které mají stejnou hodnotu. Tato hodnota je rovna 1 pro pixely, které náleží hraně. Za sousedící pixely se považují pixely, které se nachází v horizontálním a vertikálním směru (*4-connected*) nebo ještě k tomu v diagonálních směrech (*8-connected*).

Connected components vyhledáme tak, že obraz rozdělíme do horizontálních průběhů sousedících pixelů. Potom tyto průběhy označíme unikátními identifikátory, přičemž při vertikálním průběhu používáme již vytvořené identifikátory, kde to je možné. Nakonec se průběhy, které spolu sousedí spojí a označí se jednotným identifikátorem[4].

Nalezení kontur a jejich hierarchického vztahu provádí Suzuki85 následovně: Prochází vstupní binární obraz po řádcích až narazí na pixel $(i, j)$, který splňuje podmínku začátku nového ohraničení definujícího vnější nebo vnitřní ohraničení (tab. 2.1). Pokud tento pixel splňuje obě podmínky, tak se uvažuje jako bod začátku vnějšího ohraničení. Poté se mu přiřadí jedinečné pořadové identifikační číslo (NBD[5]).

| | $j-1$ | $j$ |
|---|---|---|
| $i$ | 0 | 1 |

(a)

| | $j$ | $j+1$ |
|---|---|---|
| $i$ | $>=1$ | 0 |

(b)

Tabulka 2.1: Podmínky definující začátek ohraničení pro bod $(i, j)$. (a) pro vnější ohraničení, (b) pro vnitřní

[5]NBD zkratka pro sequential number of the border

Následuje určení otcovského ohraničení. Při vertikálním skenování obrazu se ukládá vždy poslední NBD číslo nalezeného ohraničení. Toto poslední NBD bude buď otcovské ohraničení nově nalezeného anebo ohraničení, které s nově nalezeným sdílí stejného otce. To znamená, že známe NBD otce nově nalezené kontury.

Poté se postupuje po obvodu ohraničení a využívá se přitom principu hledání connected components. Odlišný je pouze systém značení, platí zde dvě podmínky:

- Pokud se aktuální pixel $(p, q)$ nachází na sledované hraně, tzn má hodnotu 1 a pixel napravo od něj $(p, q+1)$ má hodnotu 0, pak tomuto aktuálnímu pixelu přiřaď zápornou hodnotu NBD $(-NBD)$.
- Jinak přiřaď aktuálnímu pixelu $(p, q)$ hodnotu NBD, pokud se nenachází na již vysledovaném ohraničení

Tyto podmínky zajišťují, aby pixel nemohl být počátečním bodem v již vysledovaném vnitřním nebo vnějším ohraničení. Po propojení a označení celého obrysu pokračuje na další řádek a toto se opakuje až posledního řádku, kdy skončí[8].

V OpenCV je toto hledání kontur a jejich hierarchie v binárním obraze implementováno funkcí `cv::findContours()`.

#### Analýza propojených celků (segmentů)

Z nalezených segmentů v obraze lze vypočítat užitečné statistické informace, které udávají jejich vlastnosti. Tyto statistiky zahrnují výpočet:

- obsahu, neboli plochy, tj. celkový počet pixelů, které segment zabírá
- obvod, tj. součet pixelů po obvodu segmentu
- střed, neboli centroid, tj. aritmetický průměr x-ových a y-ových souřadnic vnitřních pixelů

Tyto vlastnosti lze využít pro další zpracovávání obrazu. Výpočet těchto vlastností probíhá přes obrazové momenty (*image moments*), kterými se dají vyjádřit[4][2]. V OpenCV je výpočet těchto momentů implementován pomocí `cv::moments()`.

### 2.3.4 2D Transformace perspektivy

Transformace perspektivy (*perspective/projective transformation* nebo také *homography*) je jednou ze základních transformací 2D prostoru. Zachovává rovné (přímé) linky v obraze, tj. zůstávají rovné i po transformaci. Využívá se pro namapování známých objektů v obrazu deformovaném perspektivou (kvůli úhlu a vzdálenosti) na odpovídající původní objekty, přičemž je rekonstruován jejich tvar.

Tato transformace pracuje s homogenními souřadnicemi. 2D bod, který je určen párem hodnot $X = (x, y) \in R^2$, zapsán jako sloupcový vektor (2.11),

$$X = \begin{pmatrix} x \\ y \end{pmatrix} \tag{2.11}$$

můžeme vyjádřit jako homogenní souřadnici $\tilde{X} = (\tilde{x}, \tilde{y}, \tilde{w}) \in P^2$ (při zápisu sloupcových vektorů jako $(x_1, x_2, \dots)$), kde vektory, které se liší pouze svými násobky jsou považovány za ekvivalentní. $P^2 = R^3 - (0, 0, 0)$ se nazývá 2D projekční prostor (*2D projective space*)[4].

Homogenní vektor $\tilde{x}$ může být převeden zpět do nehomogenního vektoru $x$ vydělením všech jeho složek prvkem $\tilde{w}$,

$$\tilde{X} = (\tilde{x}, \tilde{y}, \tilde{w}) = \tilde{w}(x, y, 1) = \tilde{w}\bar{X} \tag{2.12}$$

kde $\bar{X} = (x, y, 1)$ je rozšířený vektor (*augmented vector*). Homogenní body, které mají poslední prvek $\tilde{w} = 0$ se nazývají ideální body nebo také body v nekonečnu a nemají rovnocenné nehomogenní vyjádření.

Transformace perspektivy probíhá podle vzorce (2.13),

$$\tilde{x}' = \tilde{H}\tilde{x} \tag{2.13}$$

kde $\tilde{H}$ je libovolná homogenní 3 x 3 matice a nazývá se transformační matice. Výsledná souřadnice $\tilde{x}'$ musí byt normalizována, abychom dostali nehomogenní výsledek $x$ (2.14) a $y$ (2.15)[4].

$$x' = \frac{h_{00}x + h_{01}y + h_{02}}{h_{20}x + h_{21}y + h_{22}} \tag{2.14}$$

$$y' = \frac{h_{10}x + h_{11}y + h_{12}}{h_{20}x + h_{21}y + h_{22}} \tag{2.15}$$

Tato matice pro transformaci perspektivy $\tilde{H}$ se počítá ze čtyř bodů v obraze deformovaném perspektivou a k nim čtyřem odpovídajícím bodům v obraze původním a to tak, aby platil výše uvedený vzorec (2.13)[2].

Výpočet pro získaní matice pro perspektivní transformaci je v OpenCV implementován funkcí `cv::getPerspectiveTransform()`. Aplikaci matice na deformovaný obraz provádí funkce `cv::warpPerspective()`.

### 2.3.5 Kalibrace kamery

S příchodem levných kamer se rozšířila jejich dostupnost a vídáme jejich použití v čím dál více aplikacích. Jsou využívány například pro videohovory, jednoduchá videa a pro další účely. Nízká cena těchto kamer se bohužel projevuje v jejich zpracování. Kvalita optiky, která je pro běžné užívání dostatečná je nevyhovující pro využití v počítačovém vidění. Je to dáno přítomností podstatného zkreslení, které se projevuje viditelným zakřivením rovných linek v obraze snímaném kamerou. Toto zkreslení je naštěstí konstantní a tak jej lze pomocí kalibrace a následného přemapování obrazu odstranit.

OpenCV uvažuje dva typy zkreslení, radiální a tangentové. Radiální způsobuje tzv. efekt „rybího oka“ nebo „barelového vidění“. Tangentové je způsobeno tím, že součásti optiky v kameře nejsou zcela kolmé ke snímacímu členu.

Pro odstranění radiálního zkreslení se využívají vzorce (2.16) pro $x$ a (2.17) pro $y$:

$$x_{corrected} = x(1 + k_1 r^2 + k_2 r^4 + k_3 r^6) \tag{2.16}$$

$$y_{corrected} = y(1 + k_1 r^2 + k_2 r^4 + k_3 r^6) \tag{2.17}$$

Odstranění tangentového zkreslení se provádí pomocí (2.18) pro $x$ a (2.19) pro $y$:

$$x_{corrected} = x + [2p_1 xy + p_2(r^2 + 2x^2)] \tag{2.18}$$

$$y_{corrected} = y + [p_1(r^2 + 2y^2) + 2p_2 xy] \tag{2.19}$$

kde $(x_{corrected}, y_{corrected})$ jsou souřadnice bodu po odstranění zkreslení, $(x, y)$ souřadnice zkresleného bodu, $k_n$ je radiální parametr zkreslení, $p_n$ tangentový parametr zkreslení a $r^2 = x^2 + y^2$.

Tyto parametry zkreslení se uloží do jednořádkové matice (2.20):

$$Distortion_{coefficients} = (k_1 k_2 p_1 p_2 k_3) \tag{2.20}$$

Pro převod souřadnic se využívá následující rovnice (2.21):

$$\begin{bmatrix} x \\ y \\ w \end{bmatrix} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} * \begin{bmatrix} X \\ Y \\ Z \end{bmatrix} \tag{2.21}$$

Využíváme přitom homogenních souřadnic, tzn., že k vektoru 2D souřadnic přidáme složku $w$, tzn $(x, y, w)$, kde $w = Z$. $(X, Y, Z)$ jsou souřadnice bodu v 3D prostoru snímaného kamerou, jelikož je pro kalibraci využit plochý vzor, tak $Z$-ové souřadnici přiřadíme pevnou hodnotu $Z = 0$. Neznámé parametry jsou $f_x$ a $f_y$, což jsou ohniskové vzdálenosti čočky kamery $c_x$ a $c_y$, které označují optický střed. Pokud je pro obě osy použita standardní ohnisková délka s uvedeným poměrem stran $a$, pak $f_y = f_x * a$ a v uvedené rovnici budeme mít pouze jedinou ohniskovou vzdálenost $f$. Matice, která obsahuje tyto čtyři parametry se nazývá matice kamery (*camera matrix*). Parametry zkreslení jsou nezávislé na rozlišení kamery, ale $f_x, f_y, c_x$ a $c_y$ musí být přizpůsobeny aktuálnímu rozlišení, pokud se toto liší od kalibračního rozlišení.

Kalibrací kamery se nazývá proces zjištění těchto dvou matic (matice koeficientů rozostření a matice kamery). Výpočet se provádí pomocí základních geometrických rovnic a závisí na typu použitého vzoru pro kalibraci. OpenCV podporuje tři typy vzorů: černobílou šachovnici, symetrický kruhový vzor a asymetrický kruhový vzor.

Kalibrace probíhá tak, že se v obraze zachyceném kamerou vyhledá použitý kalibrační vzor. Pro každý takto nalezený vzor vzniká nová rovnice. Je potřeba najít několik těchto vzorů v různých částech obrazu snímaného kamerou, abychom dostali dostatečné množství rovnic potřebných pro správnou kalibraci. Typicky 10 a více pro šachovnicový vzor[2].

V OpenCV je začleněn celý modul zabývající se touto kalibrací – `calib3d`.

# Kapitola 3

# Popis mikrokontroléru STM32

## 3.1 Popis

Mikrokontroléry (MCU) řady STM32F103xx[9][10] v sobě zahrnují vysoce výkonné 32-bitové RISC[1] jádro ARM Cortex-M3 pracující na frekvenci až 72MHz, vysokorychlostní vestavěné paměti typu Flash (až 128 KB) a SRAM[2] (až 20 KB) a širokou škálu vstupů a výstupů, které jsou společně s periferiemi připojeny do dvou APB[3] sběrnic. Dále obsahují 12-bit analogově-digitální (ADC) převodníky, tři 16-bit časovače pro obecné použití plus jeden časovač PWM (pulsně šířková modulace). Nechybí ani standardní a rozšířené komunikační rozhraní: až dvě sériové sběrnice $I^2C$ a dvě sériové periferní rozhraní SPI, tři sériová rozhraní (synchronní/asynchronní USART), univerzální sériová sběrnice (USB) a průmyslová sběrnice CAN[4].

Zařízení pracují s napájecím napětím od 2V do 3,6V. Jsou dostupné ve dvou variantách rozsahu pracovních teplot a to buď v rozsahu od -40 až +85 °C, anebo v rozšířeném rozsahu od -40 do +105 °C.

MCU řady STM32F103xx se dodávají v 6ti různých pouzdrech, od 36 pinů do 100 pinů. Podle zvoleného pouzdra se také liší počet zahrnutých periferních zařízení. Díky těmto vlastnostem jsou mikrokontroléry řady STM32F103xx vhodné pro širokou škálu aplikací jako například: PC a herní příslušenství, GPS, tiskárny, skenery, průmyslové aplikace a další.

## 3.2 Přehled důležitých vlastností

### 3.2.1 Jádro ARM Cortex-M3

Procesor ARM Cortex-M3 je poslední generací procesorů ARM pro vestavěné systémy. Byl vyvinut pro poskytnutí nízko-nákladového řešení, které splňuje požadavky pro MCU s omezeným počtem vývodů a nízkou spotřebou při zachování velmi dobrého výkonu s pokročilým systémem reakcí na přerušení od periferií.

32-bitový RISC procesor ARM Cortex-M3 díky vyjímečné efektivitě kódu disponuje výkonem srovnatelným s 8mi a 16ti bitovými MCU.

[1] RISC je architektura procesorů s redukovanou instrukční sadou
[2] SRAM je polovodičová paměť, která k uchování svých dat nepotřebuje jejich periodickou obnovu
[3] APB je zkratka pro Advanced Peripheral Bus – sběrnice vestavěná v čipu, slouží k připojení periferií
[4] CAN značí Controller Area Network – datová sběrnice pro vzájemnou komunikaci funkčních jednotek

### 3.2.2 Vestavěná Flash paměť

MCU řady STM32F103xx obsahují vestavěnou Flash paměť o velikosti 64 nebo 128 kilobajtů, která je k dispozici pro uložení jak samotného programu, tak i dat. Tato paměť se dá přeprogramovat přes všechny dostupné rozhraní i přímo z běžící aplikace.

### 3.2.3 Pokročilý systém zpracovávání přerušení

STM32F103xx implementují Nested vectored interrupt controller (NVIC), který umožňuje zpracovávat požadavky na přerušení s nízkou prodlevou díky tomu, že je úzce svázán s procesorovým jádrem. Umožňuje řetězení požadavků na přerušení a jejich zpracovávání podle priorit. NVIC dokáže zpracovat až 43 kanálů přerušení s 16ti úrovněmi priorit.

External interrupt/event controller (EXTI) je používán pro generování požadavků přerušení nebo událostí. Skládá se z 19 vývodů, z nichž každý může být nastaven a maskován nezávisle na ostatních. Událost při které dojde ke generování přerušení se dá nastavit na vzestupnou hranu, sestupnou hranu nebo obojí.

### 3.2.4 Víceúčelové vstupy/výstupy GPIO

Víceúčelové vstupní a výstupní porty (GPIO General-Purpose Inputs/Outputs) slouží k připojení různých periferií a jejich ovládání. Všechny GPIO piny mohou být softwarově nastaveny jako výstupní v režimu push-pull (port je přes MOSFET[5] tranzistor připojen na napájecí napětí nebo na GND[6]), anebo jako open-drain (kdy je využito pouze spínání tranzistoru na GND). Případně mohou být nastaveny jako vstupní s využitím pull-up, pull-down rezistoru, nebo i bez něj jako floating (s plovoucím vstupem).

Mohou být nastaveny taktéž v režimu tzv. alternativní funkce, kdy jsou tyto piny připojeny přímo k periferiím jako jsou čítače, časovače, sériová linka a podobně. Na velkou část GPIO portů je možné připojit i analogový signál.

Na většinu z nich lze připojit jak digitální, tak analogový signál.

### 3.2.5 Časovače

Řada STM32F103xx obsahuje jeden časovač s pokročilým ovládáním (pokročilé možnosti při ladění a nastavení šířky pulzů, TIM1), tři standardní (TIM2 – TIM4), watchdog (k hlídání zacyklení programu) a systémový časovač. Tyto časovače jsou založeny na 16-bitovém automaticky nulovaném počítadle (mohou přičítat nebo odečítat) a 16-bitové předděličce kmitočtu. Dále obsahuje čtyři nezávislé kanály pro vstupní nebo výstupní porovnávání, pulzně šířkovou modulaci (PWM) nebo možnost vyslat jeden impuls na výstup.

### 3.2.6 Další důležité vlastnosti

- Univerzální synchronní/asynchronní přijímač/vysílač USART

  Hardwarová podpora řízení toku na sériové lince signály RTS (request to send) a CTS (clear to send)

- Real Time Clock modul (RTC)

  Modul pro generování reálného času

[5] MOSFET je elektrickým polem řízený tranzistor
[6] GND je zkratka pro uzemnění

# Kapitola 4

# Návrh střelnice

Smyslem laserové střelnice je poskytnutí možnosti tréninku střelby ze zbraně. Vzniká jako alternativa ke klasické střelnici se střelnými zbraněmi. Její výhodou je, že může být umístěna prakticky kdekoli, protože nevyžaduje specifické požadavky na zázemí a bezpečnost, jako u střelnice klasické, přičemž se snaží zachovávat co nejvěrohodnější zážitek ze střelby.

Využívá se upravených airsoftových[1] zbraní. Tyto zbraně jsou věrnými replikami skutečných zbraní, ale jako munici využívají malé plastové kuličky namísto střelných nábojů. Disponují dokonce i zpětným rázem při střelbě. Pro účely laserové střelnice se tyto zbraně upravují – umisťuje se na ně laser a řídící elektronika. Z pochopitelných důvodů se přestane používat kuličková munice, ale mechanický systém výstřelu je zachován, kvůli zpětnému rázu a zvuku, který produkuje. Namísto toho je střelba provedena právě tímto laserem, který se při výstřelu na krátký čas vysvítí.

Laserová střelnice se skládá z těchto částí:

- upravené airsoftové zbraně (zbraní),
- projekční plochy, na kterou je promítán terč,
- videokamery, která tento terč snímá a
- počítače, na kterém je spuštěn software na rozpoznávání zásahů.

Tyto části jsou uspořádány tak, aby nebránili volnému výhledu na terč. Využití tohoto celku nemusí být nutně jenom pro simulaci reálné střelnice, ale uplatní se např. i pro jednoduché hry založené na střelbě, kdy se na projekční plochu místo terče promítá právě tato hra. Jako příklad může sloužit hra, ve které je úkolem lovit poletující kachny nebo trenažer ozbrojených složek a další.

V této kapitole popíši návrh této laserové střelnice jak po stránce softwaru, tak po stránce hardwaru ovládajícího airsoftovou zbraň.

## 4.1 Návrh softwaru střelnice

Software virtuální střelnice vykresluje střelecký terč buďto na projekční plochu dataprojektorem nebo na LCD displej. Terč je vykreslován programově v závislosti na velikosti kreslící plochy, tj. velikosti okna a je složen z kruhů vykreslených od středu zobrazovací plochy.

[1] Airsoft je druh vojenského sportu se zbraněmi střílejícími plastové kuličky

Z USB[2] video kamery snímající tento vykreslený terč se získává obraz, ve kterém se vyhledá stopa laseru. Tento obraz se před jeho zpracováním upraví, aby stopa laseru v něm byla jednoznačně dohledatelná. Poté se přistoupí k samotnému zjištění přesného místa stopy laseru a následně se ono místo zásahu zakreslí do vykresleného terče. Bude se při tom vyžívat knihovna OpenCV.

### 4.1.1 Zjištění vztahu mezi projekční plochou a obrazem z kamery

K tomu, abychom byli schopni zjistit pozici určitého bodu na zobrazované projekční ploše (na které bude zobrazován terč) v obraze snímaném kamerou potřebujeme vědět jak tyto dva obrazy spolu souvisí. Jako výchozí podmínku si stanovíme, že kamera musí snímat projekční plochu celou a zároveň tato plocha musí vyplňovat co největší část v obraze kamery. Je potřeba zjistit projekční vztah mezi souřadným systémem tohoto obrazu a projekční plochy. Je to z toho důvodu, že kamera nebude nikdy přímo kolmá k terči, vždy k němu bude natočena pod nějakým úhlem a v určité vzdálenosti, kvůli čemuž vzniká zkreslení způsobené perspektivou. Ke zjištění transformačního vztahu použijeme kalibrační obrazec, který vykreslíme a budeme jej hledat v obraze snímaném kamerou (viz kap. 2.3.4). Po zjištění tohoto vztahu můžeme přistoupit k samotnému hledání zásahů laseru.

### 4.1.2 Detekce stopy laseru

Jelikož je intenzita světla laseru velmi vysoká, je pro běžné kamery problém rozpoznat jeho barvu – jeví se jako ostré bílé světlo s případným barevným prstencem okolo, jak intenzita na okraji klesá a tudíž může splývat s ostatními světlými částmi obrazu. Proto než přistoupíme k samotné detekci je nutné upravit čas expozice kamery tak, aby zachytávala pouze velmi jasné paprsky. Díky tomuto získáme obraz, ve kterém by měla jít velmi dobře rozpoznat stopa laseru. Zároveň pozadí na kterém laser hledáme nemusí být statické, protože světelná intenzita tohoto pozadí je mnohem menší, než intenzita laseru. Použijeme klasický, běžně dostupný, červený laser s výkonem 5mW. Z toho vyplývá, že hledaná stopa je červená a má nějaký rozptyl, to znamená, že se rozprostře po určité ploše v místě dopadu na snímanou plochu. Ke zjištění místa zásahu vypočteme střed této plochy.

### 4.1.3 Kalibrace a nastavení kamery

Jelikož web kamery vetšinou využívají levné optiky, které více nebo méně zkreslují výsledný obraz, bude implementován systém pro kalibraci kamery, který tato zkreslení dokáže odstranit (viz kap. 2.3.5). Napomůže to také k lepší kalibraci zkreslení daného perspektivou a přesnějšímu určení místa zásahu.

### 4.1.4 Grafické uživatelské rozhraní

Pro ovládání aplikace implementujeme grafické uživatelské rozhraní (GUI), které umožní její nastavení pro správnou detekci laseru, výběr a kalibraci kamery a kalibraci zkreslení způsobeného perspektivou. Dále bude obsahovat prvky pro ovládání jednoduché hry – střelnice. Ve zobrazovaném terči se bude počítat počet zásahů, jejich jednotlivé bodové ohodnocení a celkový počet bodů. Umožní nastavit omezení počtu bodů, zásahů a času, po kterém hra skončí.

[2]USB je zkratka pro Universal Serial Bus – komunikační rozhraní v počítači

## 4.2 Návrh zařízení pro ovládání zbraně

Pro ovládání airsoftové zbraně použijeme mikrokontrolér (MCU) STM32F103C8 popsaný v kapitole (3). Vytvoříme desku plošných spojů (DPS) s patřičnými vstupy a výstupy pro snímání spouště, řízení elektrického motoru zbraně a laseru. Tato DPS bude umístěna v obalu od původního zásobníku zbraně. Vytvoříme program (firmware) pro tento mikrokontrolér, kterým dosáhneme požadované funkčnosti zbraně. Upravíme zbraň tak, aby spolupracovala s MCU.

### 4.2.1 Návrh elektroniky ve zbrani

Airsoftová zbraň se skládá z elektromotoru, pružiny a pístu, který stlačeným vzduchem vystřeluje kuličky z hlavně a mechaniky, která tuto pružinu stlačuje (anglicky se tato mechanika označuje *mechbox*). Jelikož chceme zachovat tuto mechanickou funkčnost zbraně, abychom nepřišli o zpětný ráz a pocit výstřelu, musíme detekovat okamžik samotného výstřelu. Toto se provede umístěním čidla do mechaniky zbraně, indikující každý jednotlivý výstřel a to tak, že se nejprve sepne při stlačování pružiny pístu a při následném výstřelu (uvolnění pružiny) se rozepne. Díky tomu je možné zjistit, kdy se má vysvítit laser, což je ekvivalent výstřelu namísto plastové kuličky, kterou zbraň střílela původně.

Dále se musíme odpojit kohoutek spouště, který funguje jako mechanický spínač motoru a přepojit jej na vstup MCU. Je to z toho důvodu, abychom mohli výstřely ovládat programově. To s sebou přináší i nutnost ovládat motor zbraně. Ten budeme spínat pomocí MOSFET tranzistoru ovládaném z MCU. Jeho zapojení je popsáno v podkapitole o přestavění ovládání motoru zbraně (6.1).

Nakonec se na zbraň umístí laser a konektor, který je upevněn v otvoru pro zásobník a slouží k propojení DPS s MCU ovládajícím tuto zbraň.

### 4.2.2 Návrh desky plošných spojů mikrokontroléru

Při návrhu desky jsem vycházel referenčního návrhu pro mikrokontroléry řady STM32F10[11] pro základní a důležité části zapojení, které jsou nezbytné pro jeho správný chod. Jedná se o napájení, výběr a správné zapojení krystalů generujících základní kmitočet a piny pro programování MCU. U napájení je použit stabilizátor, který upravuje napětí na 3,3V vyžadované MCU a zároveň slouží jako ochrana v případě zapojení nevhodného zdroje. Tento návrh jsem pak rozšířil a upravil podle potřeb využití jako ovládací jednotky airsoftové zbraně.

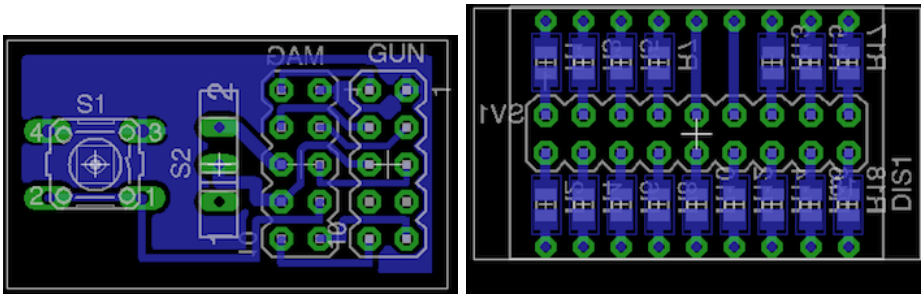
Protože jsou zapotřebí vstupy a výstupy pro ovládání jednotlivých částí, je na desce umístěn konektor pro připojení těchto částí umístěných ve zbrani. Piny z tohoto konektoru jsou připojeny na GPIO piny MCU. Jedná se o:

- Vstup kohoutku spouště zbraně, kterým se detekuje začátek výstřelu.
- Vstup ze senzoru indikujícím výstřel.
- Výstup na ovládání motoru (na MOSFET, viz kap. 4.2.1), který realizujeme pomocí optočlenu, abychom docílili galvanického oddělení výkonného motoru ve zbraní od elektroniky mikrokontroléru.
- Výstup na laser, který je ovládán bipolárním tranzistorem a to z toho důvodu, že GPIO výstupy mikrokontroléru dokáží dodávat pouze omezené, pro laser nedosta-

tečné, množství proudu. Proto je laser připojen přímo na napětí z baterie, která napájí celou DPS a je ovládán právě přes tento tranzistor.

Dále je na konektoru umístěn pin indikující zasunutí zásobníku ve zbrani a piny pro vypínač napájení DPS a pro tlačítko, kterým je simulováno přidávání nábojů do zásobníku.

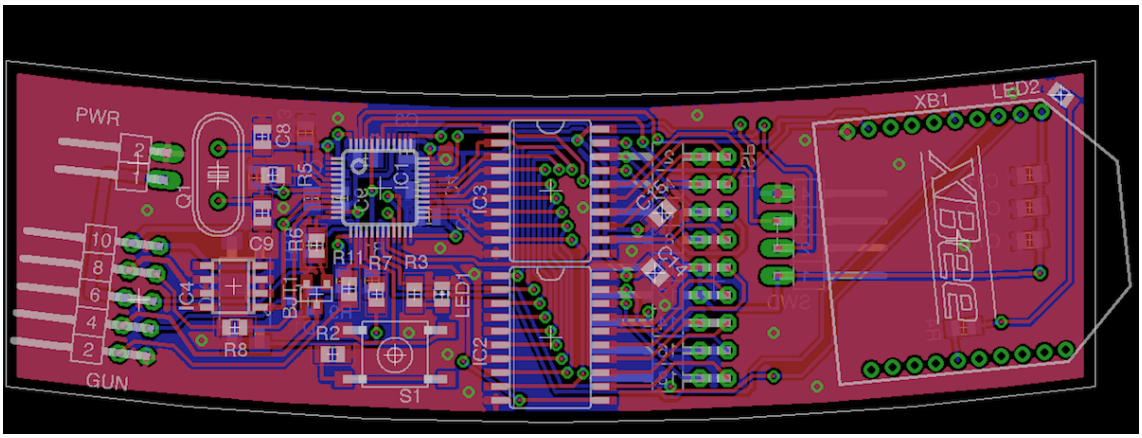
Tento konektor je připojen na pomocnou DPS umístěnou na vrchu zásobníku, která obsahuje vypínač napájení, tlačítko pro přidávání nábojů a konektor k připojení do zbraně (obr. 4.1 (a)). Druhá pomocná DPS, obsahující dva segmentové displeje, je umístěna na spodku zásobníku a slouží pro zobrazení aktuálního počtu nábojů (obr. 4.1 (b)). Tyto displeje jsou připojeny na integrované klopné obvody (IO) umožňující jejich ovládání přes jednu sadu pinů z MCU, které reprezentují jednotlivé segmenty displeje. Tyto IO si zapamatují nastavené stavy segmentů a napájí jednotlivé displeje.




Obrázek 4.1: Návrhy pomocných desek plošných spojů, (a) vlevo vrchní, (b) vpravo spodní

Všechny GPIO vstupy jsou připojeny přes ochranný rezistor na piny MCU, aby se předešlo jeho poškození v případě špatného napojení konektoru. Vstupy jsou navrženy tak, že když jsou čidla (tlačítka) rozpojené je na nich logická 1 a po jejich sepnutí přejde na hodnotu 0. Je to dáno tím, že tyto piny jsou napájené z MCU a jejich sepnutí se provede uzavřením obvodu na společnou zem MCU.

Návrh DPS MCU je znázorněn na obrázku 4.2. Celkové blokové schéma zapojení DPS zásobníku a pomocných DPS je uvedeno v příloze C.




Obrázek 4.2: Návrh desky plošných spojů mikrokontroléru

Na DPS je příprava pro připojení bezdrátového modulu, který by sloužil pro komunikaci

se softwarem střelnice, pro možnost dalšího rozšiřování funkčnosti.

### 4.2.3 Návrh firmware mikrokontroléru

Software mikrokontroléru (MCU) slouží k ovládání zbraně a jako virtuální zásobník. Při spuštění se provede potřebná inicializace jak samotného MCU, tak i využívaných GPIO vstupů a výstupů.

Vstupní piny se nastaví do výchozího režimu logické 1 (push-up režim, viz kap. 3.2.4) a indikace jejich sepnutí se děje v závislosti na druhu vstupu. Pokud jde o spoušť zbraně nebo tlačítko nabíjení, tak je jejich stisknutí definováno sestupnou hranou funkce logického obvodu, tzn. při přechodu z logické 1 do logické 0. U čidla výstřelu je aktivační hrana vzestupná a to z toho důvodu, že čidlo se nejdříve při stlačování pružiny sepne a rozepne se až při následném výstřelu zbraně. Při vstupu indikujícím, jestli je zásobník ve zbrani zasunut se uvažují hrany obě, jak sestupná, tak vzestupná a to z toho důvodu, že chceme provést akce jak při zasunuti, tak při vysunutí zásobníku.

Výstupní piny GPIO nakonfigurujeme v režimu push-pull, tj. buď poskytují proud pro dané zařízení anebo jej z něj absorbují (viz kap. 3.2.4). Pro pin, na který je napojen laser nakonfigurujeme časovač, který při aktivaci vyšle jeden puls o zvolené šířce. Tím docílíme dobu vysvícení laseru pouze na zvolený krátký okamžik.

Funkce virtuálního zásobníku spočívá v tom, že pomocí tlačítka nabíjení přičítáme počet nábojů v zásobníku a při výstřelu se tento počet odečítá. Jakmile je tato hodnota na nule, tak zbraň přestane střílet. Stav počtu nábojů se zobrazuje na připojených displejích, které jsou při inicializaci zařízení nakonfigurovány také.

# Kapitola 5

# Implementace rozpoznávacího softwaru střelnice

Základem dobře vytvořené aplikace je správný přistup k řešení problému. Z tohoto důvodu jsem použil některé velmi známé návrhové vzory (*design patterns*), které obecně řeší problémy často se vyskytující při vývoji aplikací.

Prvním z nich je tzv. **Strategy pattern**, jehož principem je, aby algoritmus byl zapouzdřen v třídě (*class*) a skryl jeho implementaci za přímočaré rozhraní. Dalším je **Singleton pattern**, který řeší přístup k instanci třídy a zajišťuje, ze tato instance bude během celého běhu programu jediná. A posledním využitým je **Controller pattern**. Tento vzor zavádí speciální třídu, pomocí které se ovládá celá aplikace a je výchozím bodem kombinujícím jednotlivé části (moduly, třídy) dohromady. V naší aplikaci je implementován pomocí Singleton vzoru.

Aplikace je založena na konceptu **Modev-View-Controller** (MVC) softwarové architektury. Smyslem této architektury je oddělit vlastní logiku aplikace od uživatelského rozhraní (GUI), využívá se přitom výše uvedených vzorů. Skládá se ze tří částí[3]:

- **Model** – obsahuje vlastní výpočetní logiku aplikace, tzn. algoritmy, metody a data ke zpracování. Při změně stavu o tom informuje Controller, který podle potřeby pošle data do View pro zobrazení, případně informuje přímo View.
- **View** – je reprezentován uživatelským rozhraním. Předkládá data uživateli ve srozumitelné formě a umožňuje uživateli aplikaci ovládat. Jakmile jsou k dispozici nová data anebo se změní stav aplikace, tak provede patřičné úpravy pro informování uživatele.
- **Controller** – zastává roli prostředníka, který propojuje Model a View. Dostává požadavky od View a přeposílá je na odpovídající metody v Modelu, čímž upravuje jeho stav. A naopak je informován o změnách stavu Modelu a tyto změny předává View, který je pak zobrazí.

V této kapitole popíši použité technologie a implementaci jednotlivých části programu tak jak bylo nastíněno v návrhu (viz kapitola 4).

## 5.1 Architektura programu

Implementace aplikace pomocí architektury MVC je následující. View je implementován pomocí zobrazovaných oken aplikace a k nim přidruženým třídám `MainWindow` a `DisplayWindow`,

zobrazujícím základní okno aplikace, respektive vykreslovací okno terče. Vykreslování je prováděno třídou `GLRenderer`, která využívá pro zobrazení OpenGL[1]. Dohromady tvoří vstupní body pro uživatelskou interakci s programem – grafické rozhraní. Model pokrývá aplikační logiku a je reprezentován třídami provádějícími zpracovávání. Jsou to:

- `CameraCalibrator` – třída pro kalibraci kamery. Vypočítá transformační matici, která je použita pro odstranění zkreslení kamery.
- `DistortionRemover` – slouží pro odstranění zkreslení z obrazu kamery za pomocí transformační matice
- `ProjectiveTransform` – třída pro zjištění projekčního vztahu mezi zobrazovaným terčem a obrazem z kamery a pro přepočet souřadnic mezi nimi
- `LaserDetector` – třída implementující detekci laseru v obraze
- `MatToQimage` – třída zajišťuje převod z datové reprezentace obrazu OpenCV (`cv::Mat`) do reprezentace Qt (`QImage`)
- `CameraControl` – rozhraní pro ovládání parametrů kamery. Implementace této třídy je závislá na operačním systému a ovladačích kamery.
- `Painter` – třída vykreslování terče, kalibračních obrazců a detekovaných zásahů laserem.

Třída `Controller` implementuje Controller aplikace. Má pouze jedinou instanci v programu. Využívá tříd Modelu, které skládá do výsledných bloků a slouží jako rozhraní pro jejich použití. Obsahuje data a nastavení důležitá pro správné provedení těchto výpočtů. Jsou zde instanciovány všechny objekty aplikační logiky programu. Ukládá nastavení aplikace do konfiguračního souboru.

Aplikace je implementována jako vícevláknová, kde vykonávání operací probíhá v samostatném výpočetním vláknu (*worker thread*), odděleném od hlavního vlákna (*main thread*, *GUI thread*), které zobrazuje GUI a umožňuje interakci uživatele s programem. Synchronizace mezi těmito vlákny je provedena pomocí systému signálu-slotů Qt frameworku (viz kap. 5.2) a pomocí výlučného přístupu ke sdíleným datům, který je realizován pomocí zámků (*mutex*) a pomocí semaforů v případě sdílených zásobníků.

Výpočetní vlákno je implementováno ve třídě `CameraPlay`, jejíž instance se nachází v Controlleru. Tato třída získává jednotlivé obrazy z kamery a aplikuje na ně metody z Modelu.

## 5.2 Použité technologie pro implementaci softwaru

Pro implementaci softwarové části střelnice je použit programovací jazyk C++ a Qt framework[12], ve kterém je vytvořena grafická nástavba pro uživatelské rozhraní. Qt zároveň rozšiřuje možnosti jazyka a přidává abstraktnější přístup k některým řešeným problémům, jako je rozdělení programu do souběžně zpracovávaných vláken a jejich komunikaci mezi sebou. Jeho výhodou je multiplatformnost, tzn. že je podporován na všech hlavních operačních systémech používaných na PC (Windows, Linux, Mac OS X). Pro zpracovávání obrazu je použita knihovna OpenCV.

[1] OpenGL je knihovna pro tvorbu a zobrazování počítačové grafiky

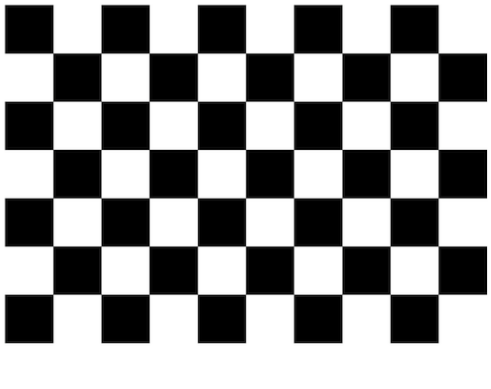
Tvorba aplikace střelnice probíhala na operačním systému Mac OS X 10.7 s verzí OpenCV 2.4.3 a verzí Qt 4.7.4.

## 5.3 Kalibrační část

Kalibrační část programu se zabývá kalibrací kamery za účelem odstranění zkreslení z obrazu získaném touto kamerou. Dále pak vyhledáním projekční plochy, na kterou se vykresluje terč a namapováním souřadnic této plochy na souřadnice plochy v obraze z kamery.

### 5.3.1 Kalibrace kamery

Kalibrací kamery rozumíme odstranění nežádoucího zkresleni z jejího obrazu (viz kap. 2.3.5). Jako kalibrační vzor je použita černobílá šachovnice (obr. 5.1). Postup je následující:



Obrázek 5.1: Kalibrační vzor[1]

Zachytíme snímek z kamery, ve kterém je vidět celý kalibrační vzor a začneme tento vzor vyhledávat. Toto provedeme pomocí funkce `cv::findCessboardPattern()`, které jako vstupní parametry předáme snímek a velikost hledaného vzoru, tj. počet vnitřních rohů, které jsou určeny čtverci po obvodu šachovnice ve svislém a vodorovném směru. Výstupem je informace, zda byl vzor nalezen a pokud ano, také body nalezených rohů mezi čtverci vzoru. Po nalezení těchto bodů jejich pozici ještě upřesníme na úrovní sub-pixelů pomocí `cv::cornerSubPix()`, která vychází z předpokladu, že každý vektor vycházející z nalezené hrany je kolmý ke změně intenzity a tudíž ke hraně. Pro uživatele nalezené rohy vykreslíme funkcí `cv::drawChessboardCorners()` (obr. 5.2 (a)) a uložíme pro pozdější zpracování. Další vyhledávání začne až po předvolené době, abychom uživateli umožnili přemístit vzor na jinou pozici. Toto opakujeme tak dlouho, dokud nezískáme dostatečný, předem zvolený počet nalezených vzorů. Abychom získali kvalitní výsledek kalibrace, je nutné tento vzor zachytit v co možná nejvíce polohách v celém snímaném záběru kamery.

Následuje samotný proces kalibrace z nasbíraných hodnot. Využívá se pro to funkce `cv::calibrateCamera()`. Vstupními parametry jsou nalezené rohy pro každý jeden kalibrační vzor a rohy šachovnice v 3D souřadnicích, které vypočteme podle zadané velikosti vzoru, kde Z-ová souřadnice se rovná nule, protože zkoumaný vzor je plochý. Výstupem funkce je 3x3 matice kamery, která je potom využita k odstranění zkreslení (obr. 5.2 (b)).

Tato kalibrace je implementována třídou `CameraCalibrator`. Uložení a pozdější načtení vypočítaných hodnot provádí třída `DistortionRemover`, která také slouží k odstranění

onoho zkreslení z obrazu z kamer. Využívá přitom právě tuto vypočtenou matici kamery.

Obrázek 5.2: (a) Vlevo průběh kalibrace, (b) Vpravo její výsledek

### 5.3.2 Projekční transformace (kalibrace terče)

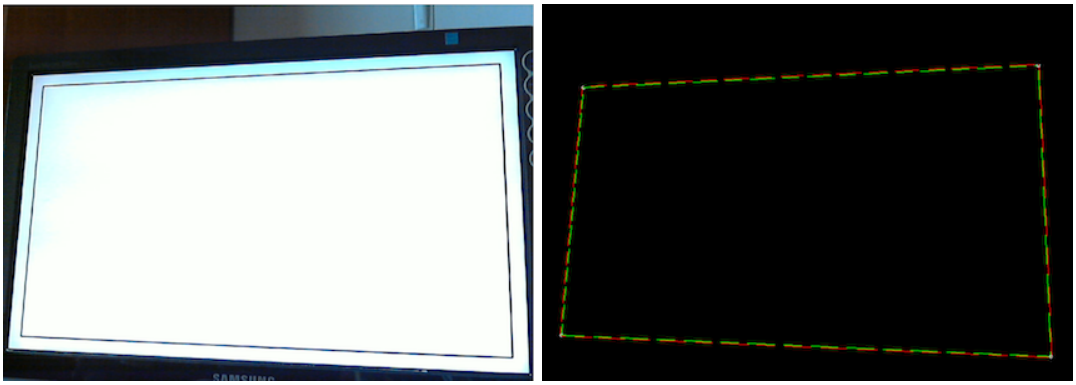
Zjištění projekční transformace vykreslovací plochy v obraze z kamery je důležité abychom mohli správně detekovat a zakreslovat místa zásahu laseru (viz kap. 2.3.4). Určení transformační matice provádíme nalezením kalibračního obrazce, který vykreslujeme na zobrazovací plochu. Kamera musí tuto plochu zabírat celou. Zároveň by snímaná plocha měla vyplňovat co největší část obrazu kamery. Kalibrační obrazec se skládá z černého obdélníku odsazeného od krajů na bílém pozadí (obr. 5.3 (a)). Implementován je třídou `ProjectiveTransform` a to následujícím způsobem:

Nejprve nalezneme kalibrační obdélník, respektive jeho čtyři rohy a to tímto způsobem: Zachycený snímek z kamery převedeme na černobílý obraz (neboli obraz intenzit černé a bíle barvy) funkcí `cv::cvtColor()`, následně aplikujeme filtr `cv::blur()`, který obraz mírně rozostří abychom se zbavili šumu v obraze. Následně vyhledáme obrysy neboli kontury. K tomu využijeme Canny algoritmus (viz kap. 2.3.2) `cv::Canny()`, kterému zadáme jakou hodnotu prahu má využít. Ten ze vstupního obrazu vytvoří binární obraz nalezených obrysů (hran). Funkcí `cv::findContours()` provedeme jejich seskupení do propojených celků, zároveň získáme jejich hierarchické uspořádání (viz kap. 2.3.3). Poté vybereme tu, která neobsahuje uvnitř sebe žádnou další (nemá žádného potomka) a je zároveň konturou největší, tj. má nejdelší obvod. Tím získáme náš kalibrační obdélník. Výpočet délky obvodu je proveden funkcí `cv::arcLength()` a počítá se z toho důvodu, že v obraze se mohou nacházet i další malé kontury, nemající žádného potomka, které byly nalezeny kvůli nežádoucího šumu v obraze.

Nyní konturu, která představuje kalibrační obdélník pomocí funkce `cv::drawContours()` vykreslíme do čistého obrazu, ve kterém budeme hledat jeho čtyři krajní rohy. Toto provádíme pomocí `cv::goodFeaturesToTrack()` implementující algoritmus Shi-Tomasi (viz kap. 2.3.2). Pro spolehlivější nalezení těchto rohů zvolíme minimální vzdálenost mezi nimi jako polovinu výšky obrazu pořizovaného kamerou, jelikož předpokládáme, ze kalibrační obrazec bude zabírat velkou plochu v tomto obraze. Následně podle souřadnic jednotlivých bodů je seskupíme v pořadí: horní levý, horní pravý, dolní pravý a dolní levý a uložíme.

Nakonec zakreslíme nalezený kalibrační obdélník i s jeho nalezenými rohy pro účely vizualizace uživateli (obr. 5.3 (b)). Je potom na něm, aby podle správnosti nebo nespráv-

nosti nalezeného vzoru upravil hodnotu prahu, který využívá algoritmus Canny na začátku tohoto zpracovávání a tím docílit lepších výsledku kalibrace.



Obrázek 5.3: (a) Vlevo obraz z kamery snímající projekční plochu s kalibračním vzorem, (b) Vpravo programem nalezený vzor

Nalezení rohů je stejně provedeno u vykreslovaného kalibračního obrazu a na základě těchto čtyř spolu korespondujících bodů je pomocí funkce `cv::getPerspectiveTransform()` vypočítána transformační matice.

Třída dále implementuje metodu pro aplikování transformace na celý obraz `applyPerspectiveTransform()` (obr. 5.4) nebo pouze transformaci jednoho bodu `transformCoordinates()`. Po úspěšné kalibraci je transformační matice uložena do souboru a načtena při dalším startu programu pro případné použití, pokud se poloha kamery a zobrazovací plochy nezmění. Přesnost transformace je ukázána na obrázku 5.4

Obrázek 5.4: Transformovaný obraz kamery

## 5.4 Hlavní část zpracovávání

Hlavní výpočetní část programu je implementována pomocí samostatného vlákna (worker thread), ve kterém se provádí veškeré operace zpracování obrazu, a které běží souběžně s hlavním vláknem aplikace (viz kap. 5.1). Toto vlákno je reprezentováno třídou `CameraPlay`, která vychází z `QThread`, což je Qt třída abstrahující práci s vlákny v programu.

Při spuštění tohoto vlákna, se volá metoda `run()`, ve které se v nekonečném cyklu získává snímek z kamery a podle nastavených parametrů se zpracovává nebo pouze posílá ke zobrazení na GUI (přes `GLRenderer`). Jednotlivé volby zpracování se nastavuji přes příslušné metody, stejně tak jako ukončení běhu vlákna. Těmito volbami jsou odstranění zkreslení způsobená kamerou (pomocí třídy `DistortionRemover`), aplikování transformace perspektivy (přes třídu `ProjectiveTransform`), detekování laseru (`LaserDetector`) a vykreslení snímku z kamery na GUI (`GLRenderer`) s možností zobrazení nalezených rohů kalibračního obrazce při kalibraci perspektivy a výstupu z detektoru laseru. Rovněž při rozpoznávání projekční transformace se z této třídy získává aktuální snímek, ve kterém se hledá kalibrační obrazec (viz sekce 5.3.2).

Nabízí se možnost nastavení maximální rychlosti zpracovávání v počtu snímků za sekundu (maxFPS[2]). Toho je docíleno počítáním doby zpracování jednoho cyklu a pokud je tato doba kratší, než doba maximální rychlosti zpracovávání ($1000/maxFPS = [ms]$), tak se před započetím dalšího cyklu vloží časová prodleva. Délka prodlevy je určena časem, který zbývá do požadovaného času zpracovávání. Pokud je čas zpracování větší, pokračuje se dalším cyklem okamžitě.

### 5.4.1 Vyhledání laseru v obraze

Vyhledání zásahu laseru na projekční ploše v obraze z kamery, která tuto plochu snímá, je stěžejní funkce programu a je implementována třídou `LaserDetector`. Dříve, než se přistoupí k samotné detekci, je na uživateli, aby nastavil expoziční čas kamery. Musí být nastaven tak, aby byl obraz co nejtmavší a pokud možno nebylo vidět pozadí plochy, ale zároveň musí být jasně vidět stopa laseru v celé této ploše.

Nastavení času expozice je v současné implementaci specifické pro systém Mac OS X při využití USB kamery podporující standard UVC[3]. Využívá převzatou třídu pro jeho ovládání `UVCCameraControl`[13]. Jelikož je napsaná v jazyce Objective-C[4], je vytvořeno rozhraní `UVCCameraCOntrolInterface` pro komunikaci se zbytkem programu, který je psán v jazyce C++. Tato třída implementuje virtuální metody rozhraní `CameraControl`. Toto rozhraní je využito pro ovládání funkcí kamery a tudíž jeho rozdílnou implementací můžeme přidat podporu pro další OS a druhy kamer.

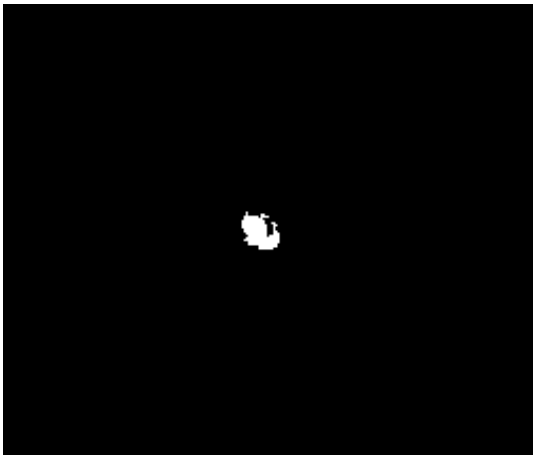
Detekce stopy laseru probíhá ve snímku získaném z kamery. Nejprve se rozdělí obraz pomocí `cv::split()` do jeho tří barevných složek, tj. R (červená), G(zelená) a B(modrá). Jelikož hledáme červený laser, tak nás zajímá intenzita v červené složce obrazu. Na tuto složku aplikujeme práh (viz kap. 2.3.1), čímž získáme binární obraz, kde je přítomna pouze stopa hledaného laseru (obr. 5.5). Velikost hodnoty prahu určuje uživatel na základě získaného obrazu tak, aby po celé projekční ploše byl laser vidět.

Poté pomocí `cv::findContours()` (viz kap. 2.3.3) vyhledáme obrysy (kontury) a jejich počet vypíšeme uživateli pro možnost vyladění nastavení detekce. Správný poměr nastavení expozičního času kamery a hodnoty pro prahování získáme tím, že počet nalezených obrysů bude roven nule, pokud v něm nebude obsažena stopa laseru a naopak bude nenulový, pokud v něm bude laser vysvícen. Stopa laseru může být detekována jako více u sebe blízkých obrysů (obr. 5.5). To je způsobeno odlesky, případně nerovnoměrnou intenzitou světla dopadajícího laseru.

[2] FPS je počet snímků za sekundu

[3] UVC je USB standard pro ovládání a získávání snímku kamery

[4] Objective-C je programovací jazyk založený na jazyku C s objektovým rozšířením převzatým z jazyka Smalltalk



Obrázek 5.5: Detail binárního obrazu stopy laseru

Následně vybereme největší obrys, ten který má nejdelší obvodovou délku, pomocí `cv::arcLength()`. Tento obrys ohraničuje hledanou stopu laseru. Nalezneme jeho střed s využitím funkce `cv::moments()`, která vypočítá obrazové momenty (*image moments*, viz kap. 2.3.3) obrysu a z nichž získáme souřadnice tohoto středu. Nakonec se nalezená souřadnice pomocí Qt signálu odešle na zakreslení do terče, které zajišťuje třida `Painter`.

Akceptování následujícího bodu zásahu se provádí až po novém vysvícení laseru. Tzn., že jakmile se stopa laseru objeví ve snímané scéně, je tato pozice brána jako bod zásahu a jeho další nepřerušený výskyt je ignorován. Další bod zásahu je zaznamenán až tento laser scénu opustí, tj., když je zachycen alespoň jeden snímek, který neobsahuje jeho stopu. Z hlediska implementace je toto provedeno tak, že alespoň v jednom snímku před akceptováním nového zásahu nesmí být nalezen obrys stopy laseru.

Při zapnuté detekci laseru se toto opakuje pro každý zachycený snímek kamery.

## 5.5 Zobrazovací část

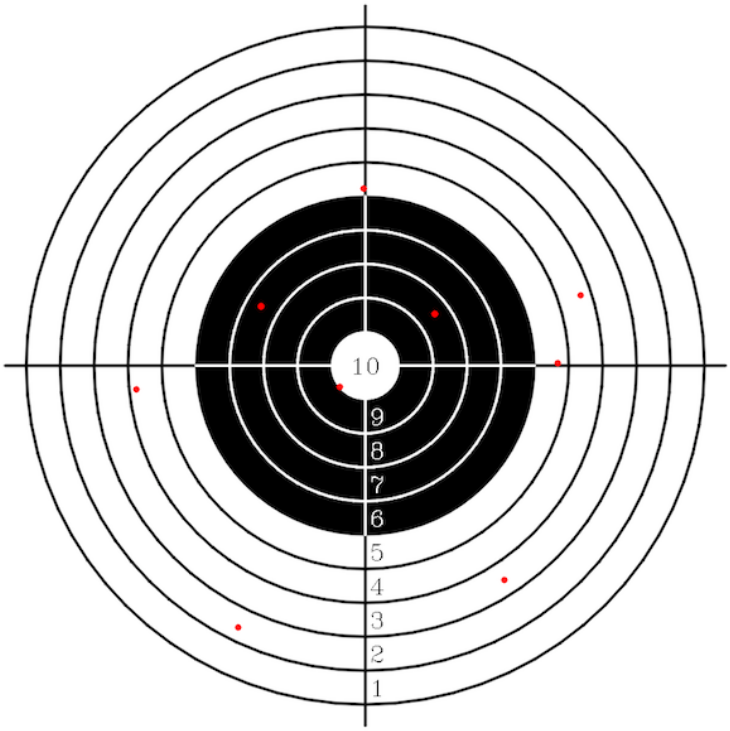
Zobrazovací část programu běží v hlavním vlákně (GUI thread). Je implementována třídou `MainWindow`, reprezentující hlavní okno aplikace a třídou `DisplayWindow` reprezentující projekční plochu – plochu na kterou je vykreslován terč a kalibrační obrazec. V těchto třídách jsou implementovány metody pro uživatelskou interakci s programem, stejně tak jako zobrazování informačních a vizuálních dat. Využívají přitom třídy `Controller`, přes kterou zasílají jednotlivé požadavky a získávají patřičné odezvy pro uživatele.

Grafické uživatelské rozhraní je implementováno pomocí Qt Widgets grafických objektů, které reprezentují ovládací a zobrazovací prvky. Soubory `Mainwindow.ui` a `DisplayWindow.ui` určují jejich uspořádání v jednotlivých oknech programu.

Kromě prvků pro konfiguraci a zobrazení video výstupu jsou zde také prvky pro ovládání a vyhodnocování jednoduché hry – bodové ohodnocení jednotlivých zásahů, celkové body, počet zásahu a čas hry.

### 5.5.1 Kreslení na projekční plochu

Vykreslování na zobrazovanou plochu provádí třída `Painter`. Implementuje OpenCV funkce pro kreslení – `cv::line()` pro kresbu přímek a `cv::circle()` pro kruhy. Třída zajišťuje vykreslení kalibračního obrazce (obr. 5.3 (a), kde je zachycen v obrazu z kamery) a terče (obr. 5.6) v závislostí na velikosti plochy, na kterou se vykresluje. Slouží k tomu metody `drawCalibImg()`, respektive `drawTarget()`. Terč je vykreslován jako kruhy se společným středem a zvětšujícími se poloměry a to tak aby bylo vyplněno co nejvíce místa v kreslící ploše. Každé jedno mezikruží je bodové ohodnoceno.




Obrázek 5.6: Terč s několika zásahy

Zároveň zakresluje body zásahu, které získává ze třídy `LaserDetector`. Tyto body jsou v souřadnicích obrazu získaném kamerou a proto musí být převedeny do souřadnic zobrazované plochy terče. K tomuto se používá metoda `transformCoordinates()` třídy `ProjectiveTransform`. Metoda přitom využívá transformační matice, která byla vypočítána při kalibraci softwaru (viz kap. 5.3.2). Z těchto souřadnic také zjišťuje bodové ohodnocení zásahu, které je potom zobrazeno uživateli.

### 5.5.2 Zobrazení grafických výstupů

O zobrazení video výstupu a nakreslených obrazců (terč, kalibrační obrazec) se stará třída `GLRenderer`, která k vykreslování používá OpenGL a vychází ze třídy `QGLWidget`. V případě videa probíhá vykreslování přes frontu snímků (*framebuffer*), která je plněna výpočetním vláknem (*worker thread*), tj. třídou `CameraPlay`.

Pomocí časovače se z této fronty ve stejně dlouhých časových intervalech vybírá snímek a následně zobrazuje. Interval časovače je nastaven na stejnou dobu, jako je nastavená

rychlost maximálního zpracovávání výpočetního vlákna. V ideálním případě je snímek který do fronty dorazí okamžitě zpracován.

Jelikož k frontě přistupují dvě vlákna, musí se zajistit výlučný přístup pouze jednoho z nich v daném okamžiku. Toto je zajištěno pomocí semaforu a zámku kolem kritické proměnné fronty. Fronta má omezenou velikost a je dodržována pomocí semaforu, ve kterém je nastaven určitý počet volných míst (zdrojů) ve frontě. Při pokusu o zápis do fronty se zabere jeden zdroj semaforu, pokud není žádný dostupný, tak se snímek zahodí. Stejně tak při pokusu o získání snímku z fronty se uvolní jeden zdroj semaforu, pokud je nějaký dostupný a vyjmutý snímek se poté vykreslí, jinak se neprovede nic. Velikost fronty je omezena na dva snímky. Je to z toho důvodu, protože zobrazení veškerých zpracovávaných snímků pro nás není důležité, pouze těch nejaktuálnějších, pokud hlavní vlákno nestíhá tyto snímky zobrazovat.

Před samotným zobrazením snímku je nutné jej převést z datové reprezentace OpenCV (`cv::Mat`, BGR) do reprezentace Qt (`QImage`, RGB). Toto provádí třída `MatToQimage` i s převodem barevného prostoru z BGR do RGB pomocí funkce `cv::cvtColor()` pro barevný snímek (viz kap. 2.3.1). Pokud se jedná o černobílý snímek intenzit, tak jej převede do RGB pomocí definované tabulky barev.

# Kapitola 6

# Stavba hardwaru a úprava zbraně

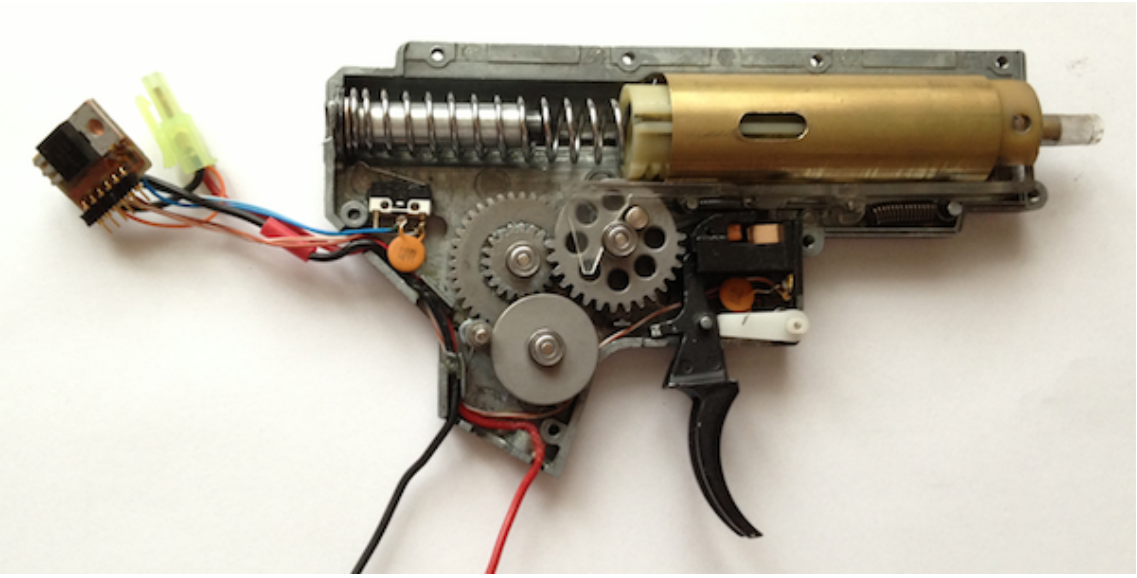
V této kapitole rozeberu úpravy provedené na airsoftové zbrani (obr. 6.1) potřebné pro dosažení požadované funkčnosti v laserové střelnici. Popíši zde vyhotovení desek plošných spojů (DPS), jejich umístění jak v zásobníku, tak i ve zbrani a funkce, které zastávají. Bude popsán způsob spolupráce jednotlivých částí k docílení funkčnosti zbraně jako celku. Pozornost bude věnována také implementaci firmwaru pro mikrokontrolér (MCU).

Obrázek 6.1: Celkový pohled na upravenou zbraň

## 6.1 Přestavění ovládání motoru zbraně

Ovládání motoru v airsoftové zbrani je řešeno mechanickým spínačem v podobě kohoutku zbraně. Při jeho stisku dojde k propojení motoru s baterií a zbraň začne střílet. Abychom mohli motor ovládat programově z MCU, je potřeba toto zapojení upravit tak, aby jej bylo možné ovládat jen velmi malým napětím. K tomu využijeme tranzistor MOSFET, na jehož elektrodu source přivedeme kladný pól baterie ve zbrani a na elektrodu drain záporný pól připojený přes motor zbraně. Při sepnutí tranzistoru se tyto dvě elektrody propojí a dojde ke spuštění motoru. Tranzistor se otevírá přivedením napětí mezi elektrodu gate a source. Elektroda gate je proto vyvedena společné s kladným pólem baterie na konektor, kterým je spojen zásobník se zbraní a v něm na optočlen, který slouží jako spínač ovládaný přes MCU. Na elektrody gate a drain je paralelně připojen rezistor pro snížení napětí přivedeného

na gate. Optočlen je použit ke galvanickému oddělení obvodu MCU od obvodu zbraně. Výsledek tohoto zapojení v mechanice zbraně je vidět na obrázku 6.2.



Obrázek 6.2: Celkový pohled na mechaniku zbraně

## 6.2 Tvorba a vestavění desek plošných spojů do zásobníku

Desku plošných spojů pro MCU vytvořenou podle návrhu uvedeném v sekci (4.2.2) a osázenou součástkami bylo potřeba umístit do zásobníku zbraně. Původní mechanickou část podávající kuličky do hlavně jsem odstranil. K samotné DPS pro MCU jsou připojeny další dvě desky, které jsou umístěny ve vrchní a spodní části zásobníku (obr. 6.3). Ve vrchní části je deska s konektorem pro připojení do zbraně a ve spodní části deska se dvěma segmentovými displeji. Bližší popis těchto desek je uveden v návrhu DPS (viz kap. 4.2.2). Propojení desek je provedeno pomocí připravených konektorů. Pro napájení MCU je použita baterie umístěna v přichystaném držáku. Aby byla celá konstrukce jednoduše vyjímatelná jsou všechny části upevněny pomocí dvou lyžin, díky čemuž ji lze pohodlně zasunout do obalu zásobníku (obr. 6.4). V tomto obalu je celá konstrukce upevněna na správném místě pomocí jednoho šroubku.

## 6.3 Elektronické osazení zbraně a kompletace se zásobníkem

Na zbraň bylo potřeba umístit modul laseru, konektor k připojení ovládací jednotky MCU, která se nachází v zásobníku, řídící modul pro elektronické ovládáni motoru zbraně a rozvést kabeláž k propojení všech částí. Popsáno bude také umístění čidla indikujícího výstřely.

Konektor k připojení ovládací jednotky v zásobníku je umístěn v otvoru pro zásobník. Jedná se o DPS s konektorem a drátovými vývody napojujícími jednotlivé piny konektoru na příslušné elektrické části zbraně, které posléze popíši. Jeden pin je určen k indikaci zasunutí zásobníku ve zbrani a je proto připojen na zem, jelikož tím je indikováno sepnutí vstupu MCU, jak je popsáno v návrhu DPS (viz kap. 4.2.2). Tento konektor je napevno přilepen


Obrázek 6.3: Pomocné desky plošných spojů umístěné v zásobníku

dvousložkovým lepidlem, aby dobře držel na svém místě. Je umístěn tak, aby jej bylo možné spojit s konektorem v zásobníku nasunutím tohoto zásobníku do otvoru k tomu určenému (obr. 6.5). Jelikož mechanizmus vkládání a uchycení zásobníku má poměrně velkou vůli, je potřeba dbát určité opatrnosti při jeho zasouvání, aby nedošlo k poškození konektorů.

Modul laseru jsem umístil na ústí hlavně do tlumiče plamene (*flash suppressor*), který je na hlaveň nasazen (obr. 6.1). Toto umístěni bylo zvoleno záměrně, protože nasazený tlumič pokračuje v ose hlavně a tudíž je pro míření ze zbraně možné používat původní mířidla. Zároveň je v něm dostatek místa pro umístění modulu laseru. Napájení laseru je je vedeno vně tlumiče do předpažbí, kde pokračuje uvnitř zbraně kolem komory hlavně až k otvoru pro zásobník, kde je napojeno na konektor pro zásobník. Je přitom využito místo pro kabeláž vedoucí od baterie k motoru, která se nachází právě v předpažbí.

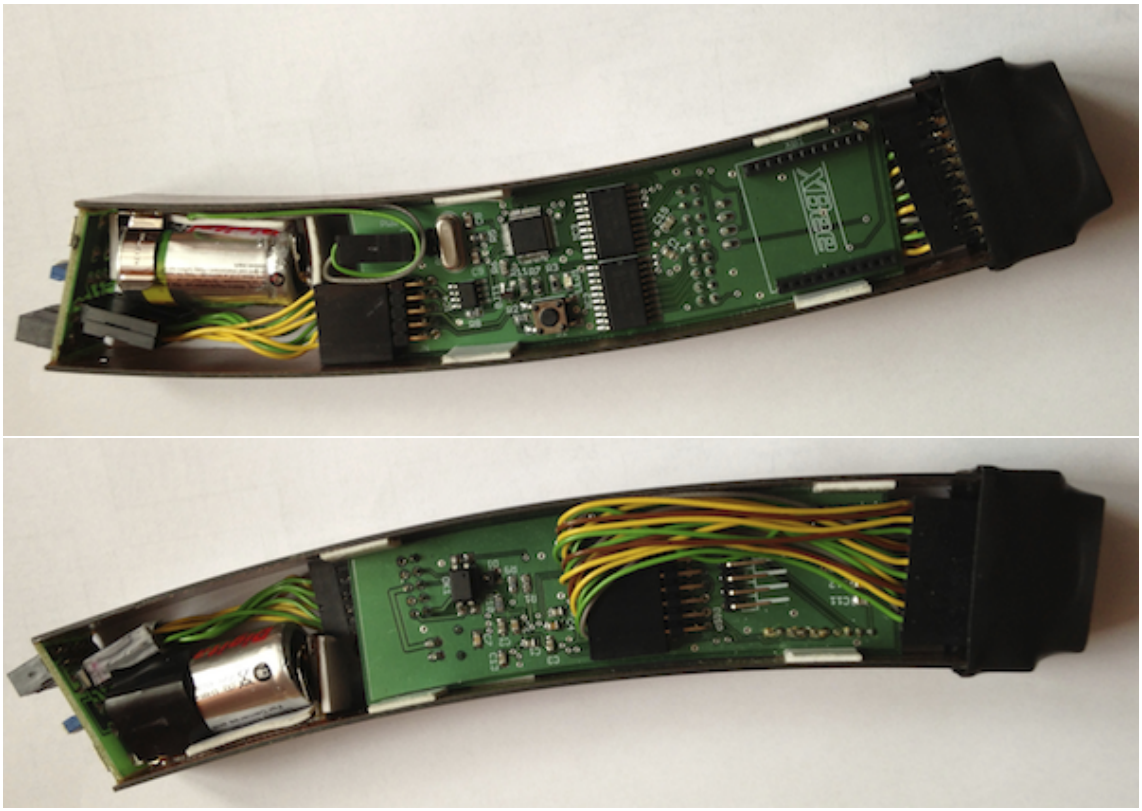
Ovládací modul motoru zbraně, jak byl popsán v části 6.1 o přestavě ovládání motoru zbraně, byl vložen do místa odnímatelné pažby za spouštěcí mechanizmus zbraně (obr. 6.6). Jelikož je tento mechanizmus vyjímatelný jsou vývody, které z modulu pokračují ke konektoru pro zásobník, odpojitelné.

Do samotné mechaniky výstřelu byl umístěn spínač, která slouží jako čidlo detekující jednotlivé výstřely. Výstřel probíhá stlačením pružiny nasazené na pístu a jejím následným uvolněním, tj. dojde ke stlačení pístu. Čidlo je přilepeno dvousložkovým lepidlem v místě kde je sepnuto, když je pružina plně stlačena a následným výstřelem se čidlo rozpojí a tímto je indikován výstřel. Čidlo je společně s vývody spouště a motoru napojeno na výše popsaný ovládací modul (obr. 6.2).

V předpažbí je vložena o jeden článek větší baterie (8-mi článková, s napětím 9.6V a kapacitě 1600mAh oproti 7-mi článkové původní), aby se tím kompenzovaly ztráty na kabeláži a ovládacím tranzistoru. Jelikož místo pro umístění baterie v předpažbí je velmi malé a i vkládání původní baterie šlo velmi obtížně, s novou baterií to bohužel jde ještě hůře a čep uchycující předpažbí nemusí jít vždy nasadit celý. Bát se o jeho vypadnutí ale nemusíme, je zde dostatečné mechanické pnutí.

## 6.4 Popis implementace firmwaru mikrokontroléru

Ovládací software pro mikrokontrolér, neboli firmware implementuje logiku pro ovládání zbraně. Je napsán v jazyce C a využívá knihovny STM32F10x Standard Peripherals Library[9] Tato knihovna poskytuje ovladače pro standardní periferie mikrokontroléru. Obsahuje funkce, datové struktury a makra pro nastavení a ovládání jak samotného jádra mikrokontroléru,



Obrázek 6.4: Vnitřní konstrukce zásobníku

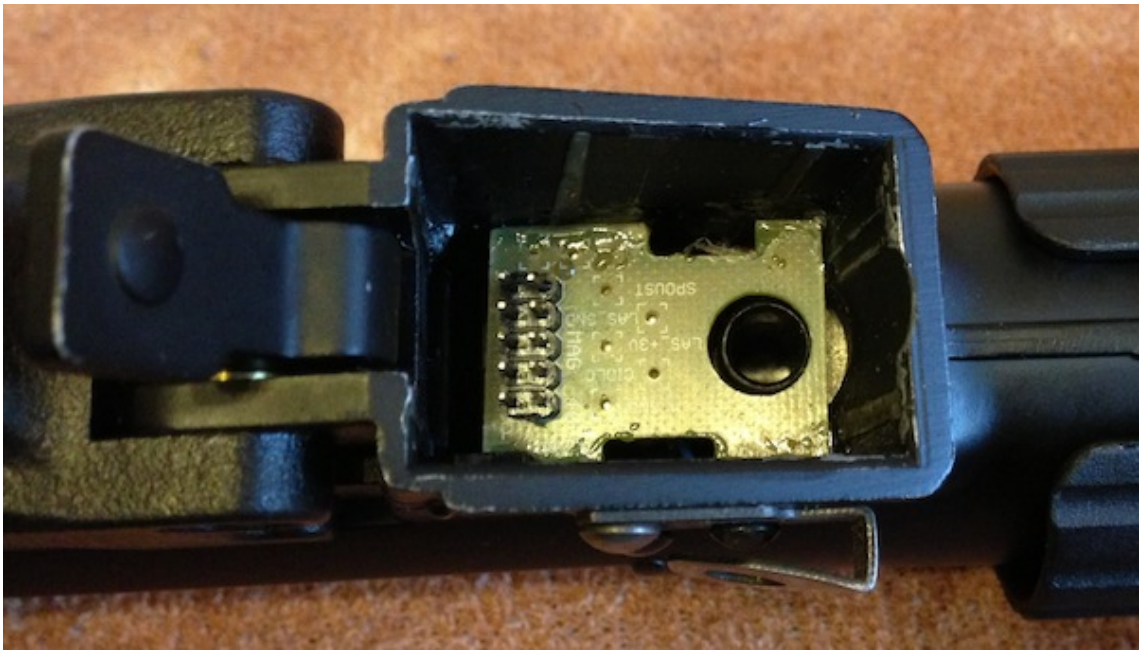
tak i těchto periferií. Slouží jako nadstavba abstrahující přímý přístup k jednotlivým programovatelným částem a tím zjednodušuje jejich nastavení a ovládání. Díky tomuto nemusíme studovat specifikace jednotlivých částí pro jejich využití v programu. Tím je urychlen vývoj a zvýšená znovupoužitelnost programu. Zároveň je kód programu přehlednější. Popisovaná implementace vychází z návrhu popsaném v kapitole o návrhu (4.2.3).

### 6.4.1 Popis částí programu

Po zapnutí napájeni se provede základní inicializace MCU, tj. počáteční nastavení ukazatele zásobníku (*stack pointer*), čítače instrukcí (*instruction pointer/program counter*), nastavení systémového kmitočtu a inicializace systémového vektoru přerušení. Je provedena spuštěním kódu v souboru `startup_stm32f10x_md_vl.s`, který je součástí použité knihovny ST, a ze kterého se pro nastavení systémového kmitočtu volá funkce `SystemInit()` ze souboru `system_stm32f10x.c`, který obsahuje toto nastavení. Po dokončení inicializace program skočí na hlavní funkci programu `main()`.

Funkce `main()` je implementována v souboru `main.c`. V této funkci se nejprve inicializují všechny periferie připojené na GPIO piny MCU a poté se rozběhne hlavní smyčka programu. Tato smyčka je prázdná, neboť všechny vstupní události, na které se reaguje jsou řešeny přes systém přerušení.

V souboru `board.c` jsou implementovány všechny funkce inicializace připojených periferií a jejich ovládání. V přidruženém hlavičkovém souboru `board.h` jsou přítomny deklarace těchto funkcí a definice jednotlivých periferií, jak jsou připojeny k MCU na DPS, pro



Obrázek 6.5: Konektor v otvoru pro zásobník

přehlednější práci s nimi.

Definice, respektive deklarace funkcí obsluhujících přerušení se nachází v souborech `stm32f10x_it.c` a `stm32f10x_it.h`. Zbývající soubor `stm32f10x_conf.h` je konfigurační soubor knihovny ST.

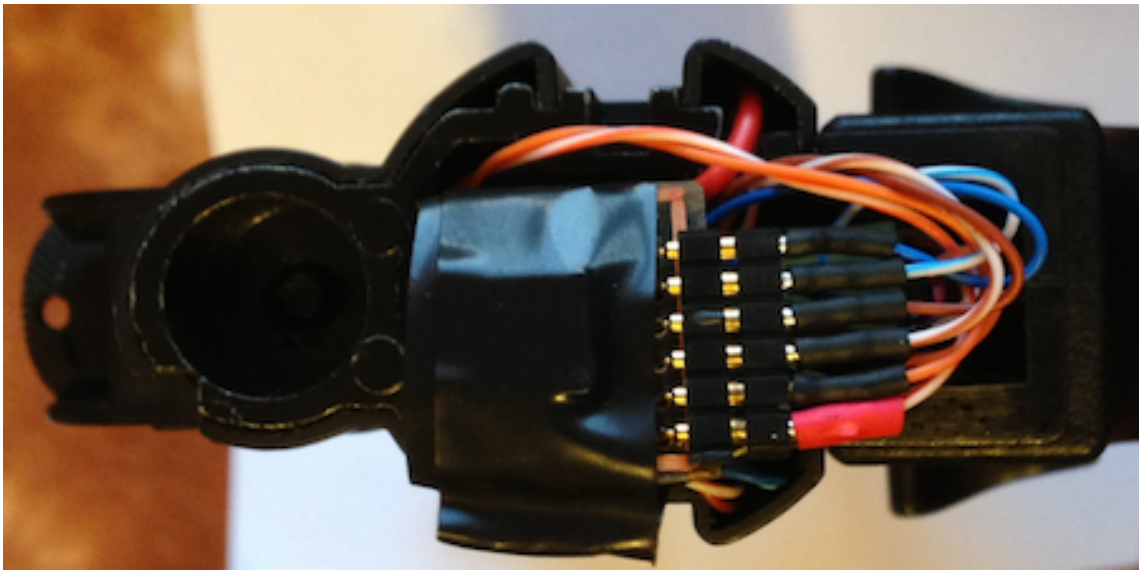
### 6.4.2 Popis implementace nastavení a ovládání periferií

Při implementaci jsem vycházel ukázkových příkladů dodávaných společně s knihovnou ST. Jednalo se o inicializaci a ovládání vstupních a výstupních GPIO pinů MCU a inicializace laseru a displejů. Veškerá implementace je v souboru `board.c`.

#### Inicializace a ovládání GPIO pinů

Inicializace vstupů a výstupů se provádí funkcí `btnInit()`. Nejprve se povolí kmitočet pro sběrnici, na které je daný pin umístěn. Poté se nastaví režim funkce pinu. Pokud se jedná o vstup, nastaví se do vstupního režimu s pull-up rezistorem. To znamená, že logická hodnota na pinu bude 1, dokud nedojde k jeho uzemnění, což značí sepnutí spínače (čidla). Pro nastavení výstupu se používá režim push-pull výstupu. Ten značí, že připojená periferie může z MCU odebírat proud. Rychlost spínání výstupu se nastaví na 50MHz. Dále se nakonfiguruje EXTI linka přerušení pro daný pin a nastaví se při jakých změnách signálu se bude přerušení generovat. Nakonec se pinu přiřadí NVIC priorita.

Ovládání těchto GPIO pinů probíhá přes funkce nastavující příslušné bity v registrech jednotlivých sběrnic, na které jsou připojeny. Jsou to funkce `on()`, `off()` a `toggle()`.



Obrázek 6.6: Umístění ovládacího modulu motoru zbraně

#### Nastavení a ovládání laseru

Laser je ovládán časovačem k jeho vysvícení pouze na krátkou dobu (50ms). Jeho inicializace se nachází ve funkci `laserInit()`. Prvně je nastaven GPIO pin do výstupního režimu alternativní funkce, kterou zastává právě časovač. Ten je nastaven do PWM režimu jednoho pulzu. Délku pulzu v milisekundách je určena periodou časovače vydělenou jeho taktem, který je získán ze systémového kmitočtu a upraven předděličkou. Tento pulz nastaví na výstupu logickou 1, kterou je aktivován laser.

#### Nastavení a ovládání displejů

Nastavení zobrazení displejů je provedeno obdobně jako laser přes časovač a je implementováno ve funkci `dispInit()`. Rozdílem je, že běží kontinuálně a v časových intervalech daných jeho periodou nastavuje hodnotu logického signálu na 0, čímž jsou rozsvíceny segmenty displeje, které jsou ovládány přes IO, jak je popsáno v návrhu DPS (4.2.2). Jinak je tato hodnota na logické 1. Tímto se dá ovládat intenzita svitu displejů. Jednotlivé segmenty displeje jsou inicializovány pomocí funkce `dispSegInit()` jako GPIO výstupy. Zároveň jsou nastaveny výstupní piny ovládající možnost zápisu do IO ovládajících displeje.

Zobrazení hodnot na displejích se provádí funkcí `setDisplaysValue()`, která postupně povolí zápis do každého ovládacího IO displejů, zapíše segmenty k vysvícení a pak tento zápis opět zakáže. O vysvícení správných segmentů se stará funkce `dispSetValue()`. Ta podle čísla, které se má zobrazit aktivuje správné segmenty.

### 6.4.3 Popis funkce programu

Veškerá funkčnost programu spočívá v zachycování externích podnětů a reakci na ně. Těmito podněty jsou vstupy z kohoutku spouště, tlačítka nabíjení, čidla detektoru výstřelů a pinu indikujícím zasunutí zásobníku ve zbrani. Zachycení těchto vstupních událostí je

řešeno pomocí přerušení. Obslužné funkce pro tato přerušení jsou implementovány v souboru `stm32f10x_it.c` jak již bylo napsáno výše.

### Vkládání nábojů

V proměnné `ammoCnt` se uchovává počet nábojů v zásobníku. Jejich přidávání se provádí přes tlačítko nabíjení umístěného na vrchu zásobníku. Reaguje se na sestupnou hranu signálu, která je vyvolána stisknutím tohoto tlačítka. Funkce obsluhující toto přerušení `EXTI0_IRQHandler()` přičte k proměnné `ammoCnt` jedničku a zobrazí stav zásobníku na displeji pomocí funkce `setDisplaysValue(ammoCnt)`. Pro odstranění záchvěvu signálu je přidána časová prodleva před samotným zjištěním, jestli je tlačítko stisknuto. Je to z toho důvodu aby nebyl jeden stisk detekován vícekrát. Počet nábojů v zásobníku je omezen na 30.

### Střelba

Vstupní pin, na který je připojena spoušť zbraně je nastaven pro zachytávání jak sestupných, tak vzestupných hran logického signálu. Při sestupném signálu je spuštěn motor zbraně a dochází ke střelbě zbraně, pokud počet ran v zásobníku je větší než nula. Toto je provedeno nastavením logické 1 na pinu ovládajícím motor. Naopak při sestupné hraně je motor vypnut. Obslužná funkce toto implementující je `EXTI4_IRQHandler()`.

Při výstřelu je čidlo v mechanice zbraně nejprve sepnuto (sestupná hrana) a poté rozepnuto (vzestupná hrana). Funkce `EXTI1_IRQHandler()` obsluhující toto přerušení reaguje právě na tuto vzestupnou hranu. Při jejím zachycení se spustí časovač laseru a laser je vysvícen na určitou dobu. Zároveň se odečte jeden náboj z `ammoCnt`, vypne se motor a vloží se časová prodleva, která reguluje kadenci zbraně. Pokud je kohoutek spouště stisknut (tzn. je nastavena střelba dávkou) a zároveň v zásobníku zbývají náboje, tak je motor znovu zapnut a pokračuje se ve střelbě, jinak se motor zbraně vypne. Regulace kadence je důležitá, protože zbraň sama o sobě střílí příliš rychle za sebou a použitá kamera, která snímá rychlostí 30fps, a systém vyhodnocování zásahů nestíhají zachycovat prodlevu mezi jednotlivými výstřely a tak nejsou některé z nich rozpoznány.

Motor se vypíná z toho důvodu, že při střelbě dávkou se musí střelba ukončit po vystřelení všech nábojů. Při nastavení zbraně na jednotlivé výstřely je vypínání motoru řešeno kohoutkem spouště, který je mechanicky ve zbrani po každém výstřelu vrácen do své původní polohy a tudíž je motor vypnut, jak bylo popsáno výše.

Časová prodleva je implementována funkcí `Delay()`, které se parametrem předává čas v milisekundách a je provedena vkládáním prázdných instrukcí (prázdný cyklus `for`). Určení délky vykonání jednoho cyklu v milisekundách je provedeno zjištěném počtu instrukcí, které zabírá zpracování tohoto cyklu a počtu instrukcí, které zvládne MCU za jednu milisekundu vykonat při nastaveném taktu procesoru. Při 24MHz se vykoná přibližně 30000 instrukcí za $1ms$ Prázdný cyklus `for` zabírá 14 instrukcí. Z tohoto se získá konstanta, kterou je požadovaný čas vynásoben a výsledek určuje počet cyklů, které se musí vykonat. Tento čas je ale pouze odhadem, protože rychlost zpracovávání instrukcí může kolísat.

### Ostatní funkce

Reakce na zasunutí, respektive vysunutí zásobníku ze zbraně je popsána funkcí `EXTI3_IRQHandler()`. Při vsunutí se vypne zobrazení aktuálního počtu nábojů v zásobníku a při vysunutí se opět zobrazí na displeji ve spodní části zásobníku.

Pro indikaci zapnutého stavu MCU je rozsvícený segment tečky na jednom z displejů. Ke kalibraci mířidel zbraně a také rozpoznávacího softwaru střelnice s laserem je připraven kalibrační mód. Aktivuje se zapnutím napájení při současném držení tlačítka nabíjení, indikován je dvěma nulami zobrazenými na displeji zásobníku. V tomto módu se nespíná motor a laser je při stisku spouště vysvícen na delší dobu (5s). Pro vrácení do normálního režimu stačí vypnout a zapnout napájení.

# Kapitola 7

# Praktické testy

V této kapitole uvedu praktické testování navrženého řešení simulátoru střelnice a jeho zhodnocení. Bude zde popsáno testovací prostředí, nastavení všech částí a samotné provedení testů. Zhodnocena bude funkčnost jak rozpoznávacího softwaru střelnice, tak i úprav provedených na zbrani.

## 7.1 Testovací prostředí střelnice

Testování softwaru střelnice se zbraní probíhalo na dvou zobrazovacích jednotkách, na LCD monitoru a na projekční ploše dataprojektoru.

### Dataprojektor

Při využití dataprojektoru se obraz promítal na projekční plátno ze vzdálenosti několika metrů tak, aby obraz vyplňoval co největší plochu plátna. Kamera snímající projekční plátno byla umístěna blíže k němu, aby jej zachycovala z co největší blízkosti, ale zároveň aby byl promítaný obraz vidět celý. Je to z důvodu, protože kamera neumožňuje zoom a čím větší plochu obrazu kamery bude zabírat, tím přesnější bude rozpoznávání míst zásahů. Kamera přitom nebyla umístěna v přímé ose s projekční plochou. Jak dataprojektor tak kamera jsou připojeny k počítači, na němž je spuštěn rozpoznávací SW střelnice. Toto prostředí je vyfoceno na obrázku 7.1.

### LCD monitor

Při testech na LCD monitoru byl projekční plochou právě tento monitor. Kamera byla umístěna do půl metru od monitoru a tento snímala. Z důvodu nedostatku místa musela kamera snímat monitor z úhlu.

V obou případech bylo okno aplikace na projekční ploše vykresleno přes celou tuto plochu a okolní světelné podmínky byly neměnné. Vývoj a testování probíhalo na operačním systému Mac OS X, CPU Intel Core 2 Duo 2.4GHz. Použita byla USB HD kamera Logitech C270.

## 7.2 Základní nastavení a způsob testování

Před přistoupením k samotným testům bylo potřeba zkalibrovat software pro zvolené rozmístění částí střelnice a zvolit jeho výchozí nastavení. Důležitá je přitom poloha kamery


Obrázek 7.1: Testovací prostředí

snímající projekční plochu.

### Nastavení softwaru střelnice

Kalibrace SW spočívá ve zjištění projekční transformace zobrazované plochy v obraze z kamery pomocí kalibračního obrazce a nastavení času expozice kamery ve správném poměru s hodnotou prahu využívaného pro detekci stopy laseru. Pro správné zvoleni těchto dvou posledních parametrů využijeme kalibrační mód zbraně, ve kterém se nespíná motor, ale pouze se vysvěcuje laser. Zvolený poměr času expozice a prahu byl takový, že v celé projekční ploše byla stopa laseru jednoznačně programem rozpoznatelná o čemž jsem se přesvědčil ze zobrazeného výstupu, který je pro tuto kalibraci určen. Výchozí rychlost zpracovávání obrazu byla nastavena na 30FPS. Celkový postup kalibrace je uveden v příloze ovládání programu (příloha A).

### Způsob testování

Testy probíhaly navolením určitého počtu nábojů v zásobníku a následnou střelbou na projekční plochu, kde byl vyobrazen terč. Podle počtu vystřelených ran a počtu rozpoznaných zásahů se hodnotila správnost nastavení jak zbraně, tak softwaru. Ke zjištění správného počtu výstřelů (vysvícení laseru) při střelbě dávkou se použila videokamera, kterou byl snímán terč, jelikož pouhým okem spočítat jejich počet bylo nemožné při rychlosti střelby

zbraně. Následně se prošel záznam snímek po snímku a spočítal se počet vysvícení laseru. Ke zjištění kadence zbraně byl použit software na analýzu zvuku.

## 7.3 Cíl a popis testů

Základním cílem testování bylo zjištění doby potřebné pro vysvícení laseru, aby byl softwarem rozpoznán zásah při určité rychlosti zpracovávání. Tato doba přímo závisí na počtu zpracovávaných snímků za sekundu, neboli po jak dlouhých časových intervalech jsou snímky kamerou zachyceny a zpracovány. Testován byl různý počet výstřelů ať už dávkou nebo po jednotlivých ranách pří různé délce výsvitu laseru, rozdílných úhlech, kadenci zbraně a rychlosti zpracování obrazu. Hodnoty délky výsvitu laseru se nastavovaly od 38ms do 80ms. Rychlost zpracovávání programu byla nastavována od 15FPS po 30FPS, což je maximální rychlost snímání kamery.

## 7.4 Zhodnocení testů

U jednotlivých výstřelů většinou nenastane problém, pokud je doba vysvícení laseru delší než doba získání jednoho snímku kamerou. Problém však nastává v případě střelby dávkou, Jelikož při této střelbě dochází k mnohonásobným výstřelům jdoucím rychle za sebou a tudíž se zkracuje doba mezi jednotlivými vysvíceními laseru.

Princip detekce výstřelů počítá s tím, že mezi každými dvěmi výstřely musí být zachycen a zpracován alespoň jeden snímek, kde není přítomna stopa laseru. Tento prázdný úsek mezi střelami dávkou ale nemusí být vždy zachycen. Důvodů může být několik a vzájemně spolu mohou souviset: Zbraň střílí příliš rychle, doba vysvícení laseru je moc krátká vzhledem k rychlosti zpracovávání snímků kamery, algoritmus detekce nestíhá zpracovávat, výpadek některých snímku kamery.

Při 30-ti FPS se každých 33ms získá nový snímek ke zpracování, takže doba svitu laseru musí být delší než těchto 33ms aby jeho stopa byla zachycena alespoň v jednom snímku. Zároveň musí být kratší než je prodleva k dalšímu výstřelu a tato doba musí být dostatečně dlouhá aby byl zaznamenán snímek bez stopy laseru. Analýzou zvuku, který zbraň vydává při střelbě jsem zjistil, že doba mezi jednotlivými výstřely v režimu dávky se pohybuje okolo 100ms. Z toho vyplývá, že 50ms pro vysvícení laseru je dobrá výchozí hodnota. Pokud se jednalo o jednotlivé výstřely, tak se toto také potvrdilo a všechny byly rozpoznány.

U střelby dávkou nastal problém, že ne všechny zásahy byly rozpoznány. Střelba zbraně byla příliš rychlá a buď doba svitu laseru krátká, že se jeho stopa neobjevila v žádném snímku kamery nebo naopak dlouhá, kdy nebyl detekován prázdný snímek a nedošlo tak k započtení dalšího zásahu. Problém způsobovalo také občasné vypadnutí snímku kamery způsobené pravděpodobně chybou v dekompresi obrazu, což jsem nemohl ověřit, jelikož se jedná o problém na straně OS, respektive ovladačů systému, připadaně funkce OpenCV pro získání snímku z kamery. Důvodem mohla být náročnost algoritmu pro detekci zásahu, jelikož při nižších počtech výstřelů za sekundu bylo toto vypadávání minimální.

Na LCD se projevil problém při střelbě pod větším úhlem od kamery, kdy intenzita stopy laseru hodně kolísala, až byla tak malá, že pro zvolenou kalibraci SW byla nedetekovatelná. Bylo to způsobeno částečným pohlcením světla laseru plochou monitoru a také jeho odrazem od této plochy.

Při využití dataprojektoru a střelbě na projekční plátno se tyto problémy neprojevovaly tak výrazně. Střelba z rozdílných úhlu neměla na výsledek žádný vliv. Rozptyl a odraz

laseru byl mnohem menší a již nepředstavoval tak velký problém. Stále zde ale byla při střelbě dávkou určitá chyba detekce. Ze 30 střel bylo zachyceno až o 9 střel méně. U LCD při střelbě pod velkým úhlem na obrazovku bylo toto číslo ještě větší. Změnou pouze doby výsvitu laseru počet nezachycených zásahů kolísal. Při 38ms se zásahy nedetekovaly spolehlivě, protože tato dobra byla příliš krátká i při rychlosti zpracovávání 30FPS. Naopak při výsvitu kolem 80ms se nestihly zaznamenat prázdné snímky a opět docházelo k chybné detekci. Spolehlivost detekce nebyla dosažena ani při zkoušení různé rychlosti zpracovávání programu. Z těchto pokusů vyplývá, že rychlost střelby dávkou je příliš vysoká pro správné zachycení všech zásahů, při použití levné USB kamery.

Tento problém jsem se rozhodl řešit programovým snížením kadence zbraně pomocí MCU, jelikož rychlost získávání snímku z kamery nemohu navýšit. Z původního počtu přibližně 10-ti střel za sekundu (kolem 100ms jeden výstřel) jsem postupným testováním došel ke spolehlivému a zároveň rychlému střílení při přibližně 5-ti ranách za sekundu (kolem 190ms). Zároveň rychlost zpracovávání snímku 30FPS se ukázala jako nejvhodnější při době výsvitu laseru 50ms. V tomto nastavení pracuje celý systém spolehlivě a detekování zásahů je správné i při střelbě dávkou.

# Kapitola 8

# Závěr

Cílem této práce bylo vytvořit simulátor střelnice, který pracuje na principu rozpoznávání zásahů laserového značkovače na zobrazovací ploše z obrazu získaného kamerou, která tuto plochu snímá a upravení airsoftové zbraně pro spolupráci s tímto systémem. Pro zpracování obrazu byla zadána opensource knihovna počítačového vidění OpenCV.

V první části jsem provedl návrh a implementaci aplikace, která na zobrazovací ploše vykresluje terč a snímá v něm vysvícený bod laseru za pomoci kamery, určuje jeho pozici a následně jej zakresluje. Pro zjištění místa zásahu se musí přepočítat souřadnice bodů zjištěných v obraze z kamery tak, aby souhlasily s body vykresleného terče kam se střílí. Program k tomuto účelu implementuje kalibraci, při které je zjištěn vztah mezi vykresleným terčem a jeho obrazem v kameře. Díky tomu není nutné mít kameru umístěnu v přesně kolmé pozici ke zobrazované ploše, tzn., že může být pod jakýmkoliv úhlem k ní. Pro správné vyhledání laseru při různých světelných podmínkách jsou připraveny volby nastavení detekce i možnost zvolení rychlosti zpracovávání snímků kamery. Zobrazovaný terč je vykreslován podle velikosti rozlišení okna s bodovým ohodnocením jednotlivých mezikruží, takže při vyšším rozlišení bude zobrazeno více kruhů a tím pádem více bodových ohodnocení. Jelikož implementovaný algoritmus detekce nevyžaduje, aby bylo pozadí zobrazovací plochy stabilní (tj. neměnné), je možné místo terče zobrazovat pohybující se scénu.

Vzhledem k tomu, že optiky levných USB kamer představují určité znatelné zkreslení, je v programu implementována kalibrace pro jejich odstranění.

U zbraně jsem navrhl, vytvořil a umístil DPS MCU do obalu od původního zásobníku nábojů. Dále jsem přidal konektory potřebné k propojení zásobníku se zbraní. Osadil zbraň laserovým značkovačem, upravil ovládání motoru zbraně tak, aby jej bylo možné řídit programově z MCU a všechny tyto části propojil. K řízeni zbraně jsem implementoval firmware pro tento MCU. Ten ovládá střelbu, detekuje jednotlivé výstřely a na jejich základě vysvěcuje laser. Dále zastává funkci klasického zásobníku, a to takovou, že do něj lze přidávat „virtuální“ náboje, podle kterých se pak určuje počet výstřelů zbraně.

Z provedených praktických testů jsem zjistil, že původní rychlost střelby dávkou byla příliš vysoká pro zachycování zásahů. Proto jsem přistoupil k programovému omezení kadence zbraně, které problém vyřešilo. Detekce bodů zásahu se ukázala být spolehlivá při správném nastavení programu vzhledem k okolnímu osvětlení, které musí být při běhu detekce neměnné.

Pro praktické využití vytvořeného systému jsem vytvořil v programu malou hru. Jedná se o počítání počtu zásahů, jejich bodového ohodnocení a celkového počtu bodů. Hru je možné spustit s časovým limitem, limitem počtu střel a/nebo počtu bodů. Předpokládané využití tohoto systému je v různých simulátorech střelnic, ale i v jednoduchých hrách.

Takovéto hry se již v praxi používají a jejich systém je principiálně podobný vytvořenému systému v této práci.

## 8.1 Možnosti rozšíření a vylepšení

Základním vylepšením by byla větší optimalizace algoritmu detekce zásahů. Tím by se snížila výpočetní náročnost, která je v tuto chvíli docela vysoká. Pro detekci většího počtu zásahů za sekundu by musela být použita kvalitnější kamera s rychlejším snímáním obrazu a program by musel být pro jejich zpracování řádně optimalizován.

Software střelnice je implementován pouze pro systém Mac OS X, i když využívá multiplatformní knihovny OpenCV a frameworku Qt. Omezení je způsobeno systémově závislým ovládáním parametrů kamery. V programu je uvedena příprava k implementaci pro ostatní OS.

Zajímavým rozšířením by byla implementace složitější hry s pohyblivou grafikou. Například lov poletujících hus, střelba na pohyblivé terče, siluety osob a další, kde by mohla být využita střelba.

Na DPS zařízení ovládajícího zbraň je nachystaná příprava pro připojeni bezdrátového modulu, díky kterému by byla možná komunikace se softwarem střelnice. Mohl by se například zobrazovat počet vystřelených a zbývajících ran v zásobníku a také zjišťovat slepé výstřely mimo plochu terče.

# Literatura


[1] WWW stránky projektu OpenCV. *Knihovna OpenCV* [online]. Dostupné na: http://opencv.org.

[2] OpenCV developer team. *Dokumentace knihovny OpenCV* [online], [cit. 1.5.2013]. Dostupné na: http://docs.opencv.org/trunk/index.html.

[3] Robert Laganiere. *OpenCV 2 Computer Vision Application Programming Cookbook*. Packt Publishing, 2011. ISBN 978-1-849513-24-1.

[4] Richard Szeliski. *Computer Vision: Algorithms and Applications*. Springer, 2010. ISBN 978-1-84882-934-3.

[5] Mubarak Shah. *Fundamentals of computer vision* [online], 1997 [cit. 1.5.2013]. Dostupné na: http://www.cs.ucf.edu/courses/cap6411/book.pdf.

[6] John Canny. A computational approach to edge detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Volume 8(Issue 6), 1986.

[7] Carlo Tomasi Jianbo Shi. Good features to track. *IEEE Conference on Computer Vision and Pattern Recognition*, June 1994.

[8] S. Suzuki and K. Abe. Topological structural analysis of digitized binary images by border following. *Computer vision, graphics, and image processing*, Volume 30, 1985.

[9] WWW stránka produktu. *Mikrokontrolér STM32F103C8* [online], [cit. 1.5.2013]. Dostupné na: http://www.st.com/internet/mcu/product/164476.jsp.

[10] Dokumentace mikrokontroléru. *Datasheet k mikrokontroléru řady STM32F103x8* [online], [cit. 1.5.2013]. Dostupné na: http://www.st.com/st-web-ui/static/active/en/resource/technical/document/datasheet/CD00161566.pdf.

[11] Dokumentace mikrokontroléru. *Getting started with STM32F10xxx hardware development* [online], [cit. 1.5.2013]. Dostupné na: http://www.st.com/st-web-ui/static/active/en/resource/technical/document/application_note/CD00164185.pdf.

[12] WWW stránky projektu Qt. *Qt framework* [online]. Dostupné na: http://qt-project.org.

[13] Dominic Szablewski. *UVC Camera Control for Mac OS X* [online], [cit. 1.5.2013]. Dostupné na: http://phoboslab.org/log/2009/07/uvc-camera-control-for-mac-os-x.

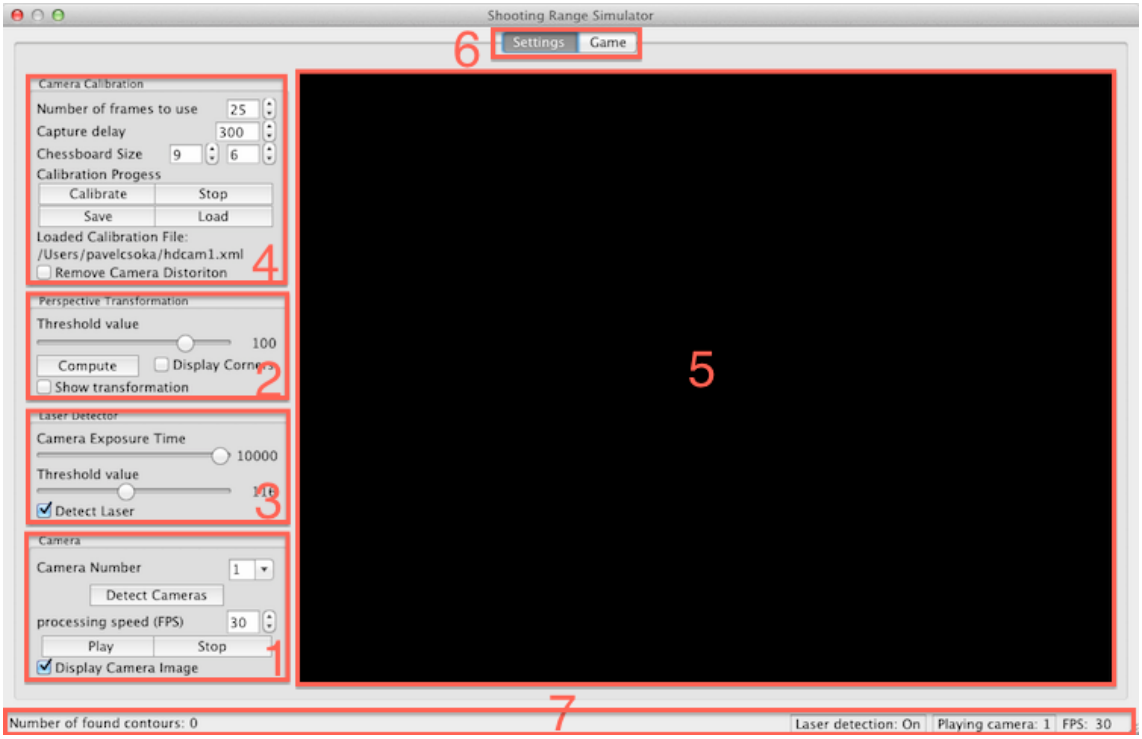
# Seznam příloh

# Příloha A

# Popis ovládání a konfigurace programu

V této příloze popíší ovládání aplikace na snímcích grafického uživatelského rozhraní a také kroky potřebné k nastavení aplikace pro správnou detekci laseru a tím pádem i zásahů. Dále zde také bude popsáno uživatelské rozhraní přichystané hry.

## A.1 Popis uživatelského rozhraní a konfigurace aplikace




Obrázek A.1: Grafické uživatelské rozhraní – Nastavení

Popis jednotlivých částí uživatelského rozhraní v pořadí uvedeném na obrázku A.1:

### 1 – Výběr a ovládání kamery

Tato část slouží pro ovládání kamery. Pomocí tlačítka „Detect Cameras“ se provádí detekce všech dostupných kamer. Výsledek vyhledávání je zobrazen ve stavovém řádku a nalezené kamery jsou vloženy do seznamu „Camera Number“, ze kterého se zvolí požadovaná kamera.

Maximální rychlost zpracovávání se nastavuje ve snímcích za sekundu (FPS) v „Processing speed“. Tlačítka „Play“ a „Stop“ slouží pro spuštění a zastavení získávání snímků z kamery. Zaškrtávacím tlačítkem „Display Camera Image“ se volí, jestli se má obraz z kamery zobrazit.

Před jakoukoli prací s programem musí být kamera spuštěná, jinak se neprovede nic.

### 2 – Transformace perspektivy

Slouží k nalezení transformační matice pro odstranění zkreslení způsobeného umístěním kamery vzhledem k projekční ploše a to pomocí kalibračního vzoru (viz kapitola 5.3.2). Při prvním pohybu posuvníku „Threshold value“ se na projekční ploše zobrazí kalibrační obrazec a provede se jeho nalezení podle navolené hodnoty posuvníku. Zároveň je na zobrazovací ploše vykreslen výsledek tohoto nálezu. Pokud není vzor nalezen správně, změnou hodnoty tohoto posuvníku docílíme správnou detekci, která závisí na daných světelných podmínkách.

Jakmile je vzor správně detekován, tak pomocí tlačítka „Compute“ spustíme kalibraci a o jejím výsledku jsem informování ve stavovém řádku. Pomocí zaškrtávacího políčka „Display corners“ můžeme zobrazit nalezený kalibrační vzor nebo obraz snímaný kamerou.

Pro ověření správnosti kalibrace použijeme zaškrtávací tlačítko „Show transformation“, kterým zobrazíme transformovaný obraz a přes něj vykreslený kalibrační vzor. Tyto vzory by se měly překrývat.

### 3 –Detekce laseru

V této části se nachází nastavení času expozice kamery („Camera Exposure Time“) a hodnoty prahu pro detekci laseru („Threshold value“) (viz kapitola 5.4.1).

Čas expozice je volen tak, aby byl obraz z kamery co nejtmavší, ale zároveň v něm byla dobře vidět stopa laseru. Po zaškrtnutí „Detect Laser“ se začíná vyhledávat laser a je zobrazen výstup této detekce. Zároveň je ve stavovém řádku zobrazen počet nalezených stop v obraze.

Hodnotu prahu upravíme tak, aby počet nalezených stop byl nulový, když na projekční ploše nebude vysvícen laser, ale zároveň musí být nenulový pokud je laser vysvícen. Toto musí platit po celé projekční ploše. Pokud se nám nedaří tohoto docílit, je vhodné upravit čas expozice kamery a zvolit jeho vhodný poměr k hodnotě prahu.

Obvykle je toto nastavení triviální.

### 4 – Kalibrace kamery

Zde se nastavují parametry kalibrace kamery (viz kapitola 5.3.1). Jedná se o:

- výběr počtu snímků použitých při kalibraci, ve kterých je rozpoznán kalibrační vzor
- nastavení prodlevy mezi přistoupením k dalšímu vyhledávání vzoru v milisekundách
- určení velikosti kalibrační šachovnice, tj. počtu vnitřních rohů, které jsou určeny čtverci po obvodu šachovnice ve svislém a vodorovném směru.

Po spuštění kalibrace se kameře ukazuje kalibrační šachovnice v různých místech a úhlech. Jakmile je získán zvolený počet snímků, je vypočtena matice kamery a kalibrace končí.

Pro ovládání jsou přítomna tlačítka: spuštění („Calibrate“) a zastavení („Stop“) kalibrace, uložení („Save“) a načtení („Load“) nalezené matice kamery do souboru. Pomocí zaškrtávacího políčka „Remove Camera Distoriton“ se voli, zda se má z obrazu odstranit zkreslení. Pro informační účel je zobrazena cesta k načtenému kalibračnímu souboru.

**5 – Zobrazovací plocha**

Na této ploše se zobrazen výstup z kamery, z detektoru laseru a kalibrace

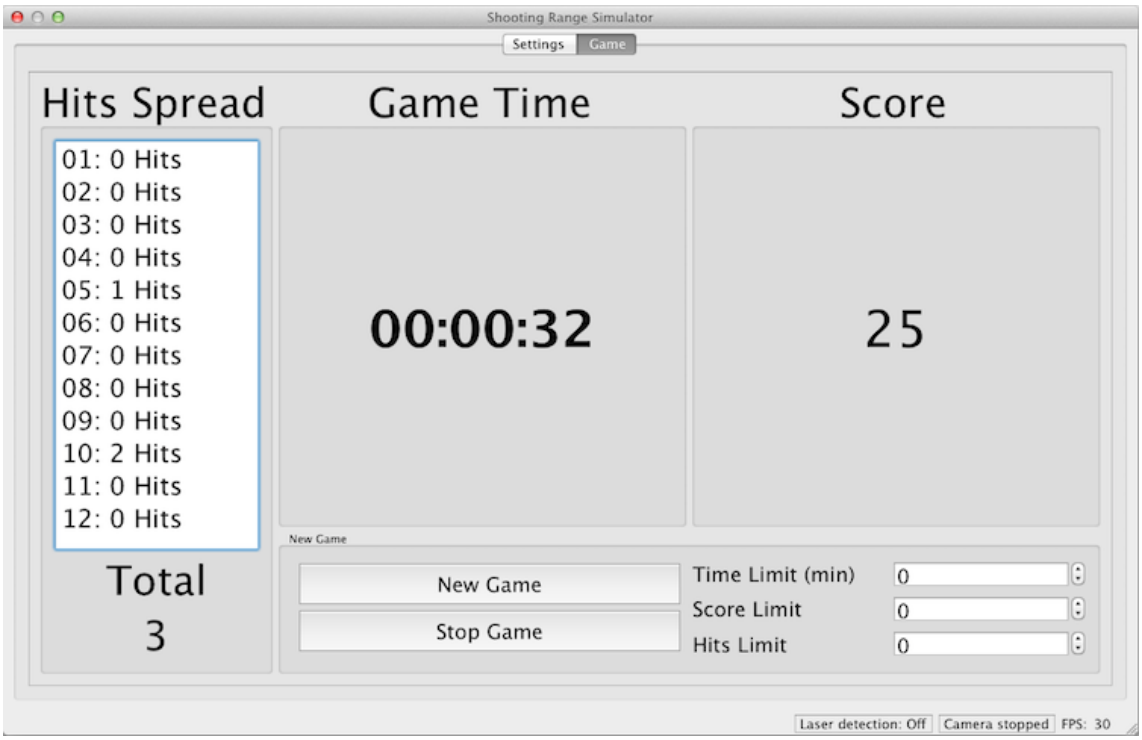
**6 – Záložky aplikace**

Slouží pro přepínaní mezi nastavením programu a ovládáním hry.

**7 –Stavový řádek**

Stavový řádek programu slouží ke zobrazení informačních zpráv. V jeho pravé části jsou zobrazeny informace o kameře, rychlosti zpracovávání ve snímcích za sekundu a indikace stavu detekování laseru (jestli je spuštěna nebo ne).

## A.2 Popis uživatelského rozhraní vestavěné hry




Obrázek A.2: Grafické uživatelské rozhraní – Hra

Vestavěná hra (obrázek A.2) se ovládá tlačítky „New Game“ pro spuštění nové hry a „Stop Game“ pro ukončení hry. Zobrazuje čas spuštěné hry („Game Time“), bodové ohodnocení jednotlivých zásahů („Hits Soread“), celkový počet bodů („Score“) a celkový počet zásahů („Total“). Nabízí možnost ukončení hry na základě zvolených parametrů, které umožňují nastavení limitu počtu střel („Hits Limit“), bodů („Score Limit“) a omezení času („Time Limit“). Nulová hodnota parametrů určují neomezený limit.

Stopy jednotlivých zásahů jsou samozřejmě vykresleny v terči.