VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INFORMAČNÍCH SYSTÉMŮ

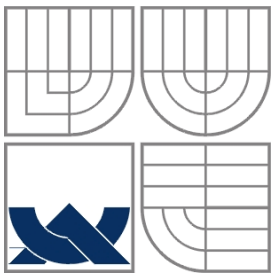
FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INFORMATION SYSTEMS

ROZŠÍŘENÁ REALITA PRO VÝUKU HRY NA KYTARU

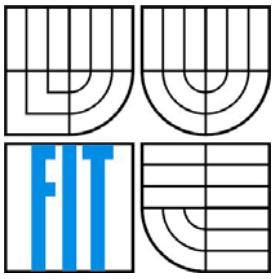
BAKALÁŘSKÁ PRÁCE
BACHELOR´S THESIS

AUTOR PRÁCE MIROSLAV KAJAN
AUTHOR

BRNO 2012



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ ÚSTAV INFORMAČNÍCH SYSTÉMŮ
FACULTY OF INFORMATION TECHNOLOGY DEPARTMENT OF INFORMATION SYSTEMS

ROZŠÍŘENÁ REALITA PRO VÝUKU HRY NA KYTARU

AUGEMENTED REALITY FOR TEACHING GUITAR PLAYING

BAKALÁŘSKÁ PRÁCE
BACHELOR´S THESIS

AUTOR PRÁCE
AUTHOR
MIROSLAV KAJAN

VEDOUCÍ PRÁCE
SUPERVISOR
ING. MAREK ŠOLONY

BRNO 2012

Abstrakt

Tato bakalářská práce pojednává o návrhu a implementaci systému pro výuku hry na kytaru prostřednictvím technologie rozšířené reality. Hráč k dosáhnutí výsledku potřebuje jen webkameru a samotný nástroj. Systém postavený na principu homografie a dalších technik pro práci s obrazem vykresluje tóny vybraného trénovaného prvku (akord nebo stupnice) přímo do snímku zachyceného webkamerou.

Abstract

This final work deals with a design and implementation of a system suitable for teaching guitar playing by using the technology of augmented reality. A webcam and the musical instrument itself are necessary for a guitar player to achieve the main aim. The system is based on a principal of homography and other techniques to work with the image. The system also renders the tones of a chosen trained element (an accord or a scale) right into the image captured by a webcam.

Klíčová slova

počítačové vidění, výuka hry na kytaru, homografie, Qt, OpenCV, webkamera, kytara

Keywords

computer vision, guitar playing teaching, homography, Qt, OpenCV, webcam, guitar

Citace

Miroslav Kajan: Rozšířená realita pro výuky hry na kytaru, bakalářská práce, Brno, FIT VUT v Brně, 2012

Rozšířená realita pro výuku hry na kytaru

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana Ing. Marka Šolonyho.

Uvedl jsem všechny literární prameny a publikace, ze kterých jsem při vypracování čerpal.

. . . . . .

Miroslav Kajan

12. května 2012

Poděkování

Tímto si dovoluji poděkovat vedoucímu bakalářské práce panu Ing. Marku Šolonymu za užitečné rady při zpracování této práce a za čas, který mi věnoval.

Obsah

1 Úvod ... 3
2 Teoretický rozbor problému ... 4
2.1 Rozšířená realita ... 4
2.2 Potřebné prostředky a technologie ... 4
2.2.1 Webkamera ... 5
2.3 Přehled podobných projektů ... 5
2.3.1 Guitar Hero ... 5
2.3.2 Neinteraktivní výukový software ... 6
2.3.3 Vědecké práce s podobnou tématikou ... 6
2.4 Cíloví uživatelé a používaný hudební nástroj ... 6
3 Návrh implementace ... 7
3.1 Princip detekce hmatníku ... 7
3.1.1 Houghova transformace ... 7
3.1.2 Detekce barevných markerů ... 9
3.2 Homografie ... 9
3.2.1 Výpočet matice pro homografii ... 10
3.3 Princip rozřazení pražců ... 11
4 Implementační prvky ... 13
4.1 Objektový návrh aplikace ... 13
4.1.1 Hierarchie tříd ... 13
4.1.2 Třída Settings ... 13
4.1.3 Třída Fingerboard ... 14
4.1.4 Třída CaptureVideo ... 18
4.2 Reprezentace akordu a stupnice ... 19
4.3 Princip mapování tónů ... 19
4.4 Grafické uživatelské rozhraní ... 19
4.5 Vývojové prostředí ... 21
4.6 Použité knihovny ... 21
4.6.1 QT ... 21
4.6.2 OpenCV ... 22
4.6.3 ARToolkit ... 22

4.7 Implementační zajímavosti 22
4.7.1 Správa paměti v OpenCV 22
4.7.3 Funkce pro získání středního momentu 23
4.7.4 Reprezentace barvy v OpenCV a Qt 23
5 Testování 25
5.1 Snímací zařízení 25
5.2 Testovací prostředí 25
6 Závěr 28
A Obsah CD 31

1 Úvod

Počítačový software na přelomu prvního desetiletí jednadvacátého století nesklízí úspěch pouze ve vodách kancelářských balíků a herních titulů. Stále více je možné vidět za určitým typem programů náhradu za učebnici či přímo reálného učitele. Toto softwarové odvětví má samozřejmě velký rozptyl – od jednoduchých testovacích programů přes elektronické interaktivní učebnice až po robustní programy, kde velká znalostní báze skutečného lektora leckdy dokáže nahradit. Přibližme se nyní tématu této práce a zaměřme svou pozornost konkrétně na lektorské programy týkající se hry na pravděpodobně světově nejrozšířenější hudební nástroj – kytaru.

Pokud nahlédneme mezi dostupný software, který v jistém slova smyslu nahrazuje funkci reálného učitele hry na kytaru a navíc ještě tuto činnost provádí netradičním a inovativním způsobem, setkáme se s velmi slabou tržní nabídkou. Softwarový trh sice nabízí produkty pro vizuální výuku hry na výše zmíněný strunný nástroj, nezachází však příliš do hloubky a nedává prostor zábavnému segmentu aplikace. Ve výsledku tak nahlížíme na směs několika produktů, které nabízejí sice pokročilou teorii, ale vše je zde prezentováno formou nevýraznou až nudnou.

Cílem této práce je tak snaha o vytvoření nevšední učební pomůcky, která staví na efektu pionýrství v oblasti spojení hry na strunný nástroj, pokročilé snímací techniky a principu rozšířené reality. Pomocí webkamery má žák možnost získat přehled o základních stupnicích a akordech netradiční formou, kdy se na obraze získaném kamerou bude za pomoci rozšířené reality zobrazovat posloupnost tónů.

Technická zpráva je dělena z hlediska rozdělení kapitol – v následující kapitole rozeberu teoretický náhled na projekt, ukáži projekty na podobné bázi, které mi v jistém slova smyslu dodaly inspiraci. Dále zde rozebírám skupinu cílových uživatelů, stručný a obecný popis programu jako celku a náležitosti, které uživatel k programu z hlediska materiálního vybavení bude potřebovat. V kapitole č. 3 popisuji kroky při návrhu aplikace, rozebírám projekt ze stránky exaktních věd s ohledem na detailnější rozebrání teorie. Kapitola č. 4 se soustředí na samotnou implementaci aplikace, popis objektového řešení, návrh tříd a obsahuje stručné úryvky zdrojového kódu. V kapitole č. 5 pak uvádím v krátkosti poznatky z fáze testování programu.

2 Teoretický rozbor problému

V této kapitole uvádím teoretický popis jednotlivých částí projektu z obecného hlediska. Kapitola popisuje základy principů rozšířené reality, popis nutných prostředků a použitých technologií. Rovněž obsahuje informace o projektech podobného ražení, které mě v rámci řešení inspirovaly a poskytly určitou znalostní bázi pro pochopení některých problémů.

2.1 Rozšířená realita

Rozšířená realita (angl. *Augmented Reality*) je adaptace reálného prostředí, jehož elementy jsou rozšířeny o počítačem generované vstupy, jako je zvuk, video, grafika nebo GPS data [3]. Rovněž bývá tento pojem někdy představován pod slovy „zprostředkovaná realita“.

V rámci tohoto projektu prostupuje rozšířená realita celým konceptem. Na snímku získaném z webkamery jsou počítačově vykreslovány body (popř. hrany detekovaného hmatníku), které dávají vůči prostoru smysl (žádné body se nezobrazí, pokud program nedetekuje kytaru).

2.2 Potřebné prostředky a technologie

Aplikace v prvé řadě staví na faktu, že uživatel nepotřebuje speciální snímací zařízení umístěné na kytaře, které se u podobných typů aplikací (viz kapitola 2.2) používá. Uživatel sám tak pro správné využití potřebuje jen nástroj (akustická či elektrická šestistrunná kytara) a dvě značky, které identifikují hranice hmatníku (nultý a poslední pražec). Tuto situaci popisuje obrázek 2.1:

Obrázek 2.1: Ukázka požadovaného umístění značek na kytaře

Vzhledem k tomu, že kytara je v drtivé většině případů jednobarevná, má uživatel možnost vybrat si z několika barev markerů (např. červená, zelená, žlutá). Tyto barvy jsou dostatečně výrazné a jejich detekce je nejrychlejší a nejpřesnější. Z hlediska technologického postačuje ke správné funkčnosti aplikace klasická webkamera (konkrétní ukázky viz kapitola 5.2). Vzhledem k robustnosti aplikace je ale doporučeno disponovat dobrým základem operační paměti (cca 1 GB) a silnějším procesorem.

2.2.1 Webkamera

Webkamera samozřejmě musí splňovat několik technických podmínek. V prvé řadě se jedná o dostatečný počet snímků za sekundu. O podstatný faktor se jedná zejména s ohledem na znatelně rychlejší detekci hmatníku a následné zobrazování tónů, rychlejší reakci na nepřesnosti vzniklé příliš složitým pozadím za uživatelem apod. Z hlediska rozlišení kamery byla primárně použita integrovaná notebooková webkamera s rozlišením 640x480 pixelů. V neposlední řadě je důležité myslet na kompatibilitu tohoto technického prostředku se softwarem (zejména s knihovnou pro práci s obrazem), v dnešní době tuto vlastnost splňuje naprosto drtivá většina na trhu dostupných webkamer.

2.3 Přehled podobných projektů

V této subkapitole uvádím několik projektu, které mi byly při práci na popisovaném programu inspirací. Žádný z nich ovšem neselektuje svůj zájem do spojení výuky hry na nástroj s interaktivním přístupem nebo není dostatečně transparentní. Dá se nicméně říci, že složením principů z jednotlivých projektů se mi podařilo tento projekt jako celek vytvořit.

2.3.1 Guitar Hero

Guitar Hero je série interaktivních, hudebních videoher, jež začíná prvním dílem s názvem *Guitar Hero*, který byl vyvinut společností *Harmonix Music Systems* a byl distribuován firmou *Red Octane* [10]. Důležitým prostředkem pro hraní je plastový ovladač tvarem připomínající právě kytaru. Hráč se snaží dosáhnout co nejvyššího skóre. Jedná se o projekt zhruba pět let starý.

Hráč získává body, pokud správně "zahraje" všechny tóny, které mu během přehrávané skladby hra předkládá. Hra na ovladači však není příliš podobná hře na reálnou kytaru: Na symbolickém hmatníku se nachází pět barevně odlišených tlačítek, které hráč drží podobně jako akordy. Přidanou hodnotou oproti rozebíranému projektu je nutnost „vyklepávání“ rytmu na podložku, která má v jistém slova smyslu funkci trsátka. Mezi další vlastnosti rozšiřující hru patří vibrapáka nebo tlačítko Star Power [10].

Podstatnou vlastností pro srovnání tohoto produktu s navrhovaným projektem byl fakt, že popisovaný produkt se těší velké oblibě, a to z mnoha různých důvodů. Rozhodně je velice interaktivní, po technické stránce precizně vyřešené, zabaví až 4 hráče najednou, navíc při poslechu vyvážené sbírky hudby od těch nejslavnějších rockových muzikantů. Setkává se ale i s kritikou, že

odvádí mladé talenty od zájmu o opravdový nástroj, a tím škodí hudbě. Což se stalo nejpodstatnějším záporem při myšlence na orientaci tímto směrem v rámci návrhu mého projektu.

2.3.2 Neinteraktivní výukový software

Na trhu existuje celá řada programů pro výuku hry na kytaru či baskytaru, ale tyto jsou v naprosto drtivé většině neinteraktivní a nabízejí uživateli možnost jen prohlížet obrázky s vykreslenými akordy či stupnicemi. Některé z nich jsou koncipovány jako klasické webové stránky či aplikace, které navíc příliš profesionálně nevypadají a jsou tvořeny "na koleni". Díky obsaženým zvukovým nahrávkám si může uživatel alespoň daný akord/stupnici "přehrát". Jedná se o jiný přístup k výukové tématice, který je sice jednoduchý, ale neposkytuje žádný faktor reálna, díky němuž by měl uživatel z procesu učení zajímavější zážitek.

2.3.3 Vědecké práce s podobnou tématikou

Patrně nejbližší přístup k podobnému typu projektů dosahuje několik vědeckých prací, které však celkovou práci nezaštiťují jako program pro výuku na hudební nástroj, ale spíše se zaobírají konkrétnějším specifikem. Například projekt [11] z brazilské univerzity Instituto Nacional De Matematica Pura E Aplicada v Brasilii se zevrubně zabývá podrobnějším studiem detekce hmatníku a jeho následnou segmentací. Jiný výzkumný projekt [12] pak řeší problém detekce hmatníku za pomoci znalostí o infračerveném světle a použití dosti robustní snímací technologie. Používají sice podobné softwarové prostředky k dosažení kýžených výsledků jako rozebíraný projekt, ale nikterak se nesoustředí na předmět výuky.

2.4 Cíloví uživatelé a používaný hudební nástroj

Cíloví uživatelé jsou lidé, kteří si přejí naučit se základní akordy a stupnice na strunný nástroj, chtějí, aby toto učení bylo prováděno zábavnou a jednoduchou formou. Vlastní aplikace je navržena s orientací na rychlý běh a svižné poskytnutí očekáváné funkčnosti. Operační systém MS Windows dobře spolupracuje se všemi programovými knihovnami (viz kapitola 4.5) a především je primárně používaným operačním systémem, nepociťoval jsem proto nutnost tvořit aplikaci multiplatformní (ačkoli vzhledem k politice využitých knihoven nebude portování aplikace větším problémem). Tudíž je program vhodný pro uživatele tohoto operačního systému.

Z hlediska nutnosti hudebního nástroje není program nikterak specificky vymezený. Jedná se o libovolný strunný nástroj s 22-24 pražci; kytara může být akustická či elektrická, nikterak nezáleží na značce, jen by délka menzury měla vyhovovat typům, pro který je aplikace navržena (viz kapitola 3.5).

3 Návrh implementace

Vzhledem k složitosti aplikace bude implementačně rozdělená do několika částí, z nichž nejdůležitější je část, která implementuje samotnou práci s obrazem.

Obraz získaný z kamery bude program zpracovávat po snímcích a v rámci detailního rozpoznání kytary detekovat podle několika nezbytných funkcí jednotlivé části nástroje. Tato část je v návrhu i v samotné implementaci tou nejpodstatnější. Navazující částí projektu je pak korektní vykreslování bodů značících tóny aktuálně cvičené stupnice nebo melodie.

Jako takový nás na kytaře zajímá hmatník, který bude nutno v každém snímku identifikovat a přesně určit jeho hranice. Řešení jak nalézt vertikální a horizontální hranice hmatníku je nespočet (např. pomocí kombinace funkce pro detekci hran a Sobelova operátoru, popř. Houghovy transformace), ale tento projekt si mimo vytvoření funkční aplikace sekundárně kladl za cíl testovat různé způsoby detekce částí obrazu, (v zájmu výzkumu) a proto pro každou část komplexní detekce hmatníku využívám jiný přístup.

3.1 Princip detekce hmatníku

Prvotním faktorem pro správné zobrazování tónů na hmatníku kytary je jeho úspěšná detekce. Ta spočívá v nalezení jeho obdélníkového obrysu v obraze. Z principu počítačové detekce obrazu v tomto konkrétním projektu kloubím dvě rozdílné metody pro nalezení specifických objektů v obraze. Konkrétně se jedná o nalezení horizontálních hranic hmatníku (tj. jeho “délky”) pomocí implementace Houghovy transofrmace (viz kapitola 3.1.1). Druhou metodou je detekce barevných markerů (viz kapitola 3.1.2), umístěných na obou vertikálních koncích hmatníku (tj. na hlavě a na krku kytary).

Kombinace zvolených způsobů byla zvolena především ze zájmu o rozmanitější implementaci a studijní zájem o zpracování těchto dvou odlišných technik a za druhé s ohledem na přílišnou nenáročnost na samotného uživatele – ten potřebuje pouze dva kusy různě barevných papírů nalepených na koncích hmatníku. Nejsou tedy vyžadovány žádné složitě sehnatelné materiály jako v případě již existujících projektů s podobným zaměřením (viz kapitola 2.3).

3.1.1 Houghova transformace

Houghova transformace je metoda pro nalezení parametrického popisu objektů v obraze. Při implementaci je třeba znát analytický popis tvaru hledaného objektu. Proto je tato metoda používána pro detekci jednoduchých objektů v obraze jakou jsou přímky, kružnice, elipsy a podobně [6].

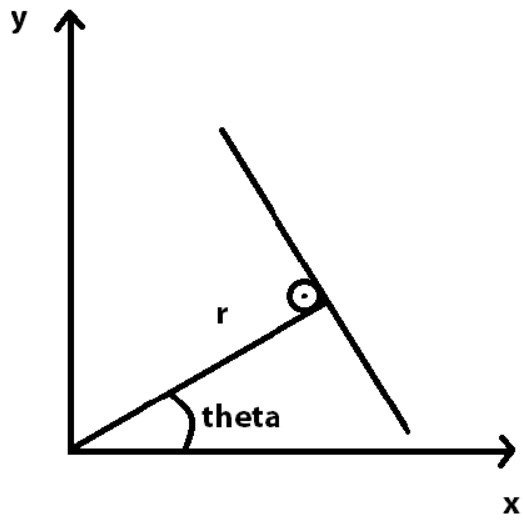
V této práci budeme Houghovu transformaci využívat k hledání dvou vodorovných přímek, které budou znázorňovat horizontální hrany hmatníku kytary. Segmentací těchto dvou přímek získáme informaci o šířce hmatníku a zároveň horizontální hranice, za které už body nebudeme

vykreslovat. Hlavní výhodou této metody je robustnost vůči nepravidelnostem a porušení hledané křivky.

Pracujeme s popisem přímky ve tvaru, který je popsán ve vzorci 1:

$$xcos\theta + ysin\theta = r \quad [1]$$

kde *x, y* jsou souřadnice námi zvoleného bodu a hledáme proměnné *r* (kolmá vzdálenost od souřadnicového středu k přímce) a θ (úhel mezi kolmicí a souřadnicovou osou x). Obrázek 3.1 znázorňuje tuto situaci:



Obrázek 3.1: *Popis hledané přímky (parametrický přístup)*

Pro pochopení je nutné si uvědomit, že body ve zpracovávaném obrázku jsou reprezentovány sinusoidami ve výše naznačeném parametrizovaném prostoru, zatímco body v parametrizovaném prostoru reprezentují čáry v obrázku. Sinusoidy se v parametrizovaném prostoru protínají ve speciálním bodu – což je v klasickém Euklidovském prostoru hledaná přímka.

Samotná metoda pracuje následovně:

1. v prvním kroku je Houghův prostor diskretizován
2. každý element tohoto prostoru, který nazýváme akumulační buňka je vynulován
3. každý bod (x_i,y_i) je transformován do diskretizované křivky (r, theta)
4. hodnota akumulačních buňek podél této křivky je inkrementována
5. souřadnice maxima v akumulační rovině (r_{max}, $theta_{max}$) jsou parametry hledané přímky v obraze

Z hlediska využívání grafické knihovny OpenCV (viz blíže kapitola 4.6) bylo vhodné pracovat s vestavěnými funkcemi, u kterých je možné si nastavit různá specifika jako je například počet

nalezených čar či kritéria akceptování čáry jako výstupu. Implementace musí splňovat následující kroky pro nalezení dvou konkrétních přímek. Přímky jsou:

a) rovnoběžné (vzhledem k různým typům kytar se pracuje s lehkou tolerancí rovnoběžnosti)
b) nesvislé
c) a mají od sebe v rámci obrazu maximální vzdálenost (nepředpokládá se, že se v obrazu objeví široké vertikální objekty ohraničené dvěma rovnoběžnými čarami), je však vlastní.

3.1.2 Detekce barevných markerů

Další oblastí, kterou projekt implementuje, je trasování pohybu dvou barevných objektů, které budou na nástroji ze stran umístěny. Toto řešení bylo zvoleno proto, že ať již klasická nebo elektrická kytara nemá žádné signifikantní body pro konec a začátek hmatníku z hlediska vertikálního. Naštěstí je detekce specifické barvy velmi nenáročnou a zároveň účinnou formou jak vyhrazenou oblast detekovat. Postup detekce je následující:

- Získáme snímek z kamery
- Pomocí funkce tresholdu se v HSV modelu zaměříme na získání pouze dané barvy
- Aktuální pozici ukládáme do struktury reprezentující dvojici bodů

3.2 Homografie

Pro rámec této práce se stává naprosto nejpodstatnějším pojmem výraz homografie. V oblasti počítačového vidění se dá tzv. *planární homografie* definovat jako projektivní mapování z plochy R1 na plochu R2, u kterého nezáleží na vzdálenosti mapovaného bodu od kamery. Z hlediska matematického se pak dá tato definice popsat vztahem (viz vzorec 2):

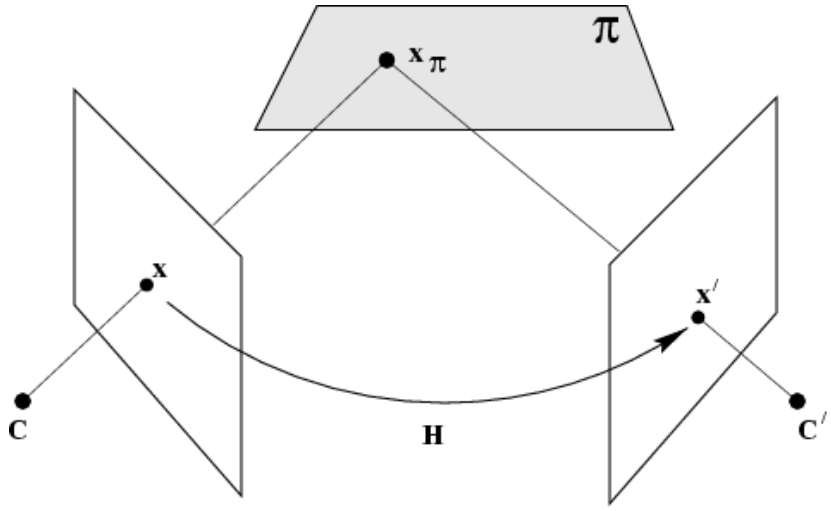
$$s_i \begin{bmatrix} x'_i \\ y'_i \\ 1 \end{bmatrix} \sim H * \begin{bmatrix} x_i \\ y_i \\ 1 \end{bmatrix} \qquad [2]$$

Graficky je princip homografie znázorněn na obrázku 3.3, kde homografie popisuje mapování bodu *x* z roviny *C* na rovinu *C'*. V rámci tohoto projektu je výpočet homografie nezbytný k zachycení aktuální pozice hmatníku a korektnímu vkládání tónu do samotného sejmutého snímku z webkamery. Je nutné upozornit na fakt, že homografie může být mezi dvěma zachycenými snímky pouze a pouze tehdy, když [8]:

- Oba snímky popisují stejnou rovinu
- Oba snímky pocházejí ze stejného zařízení pro zachycení, liší se jen jeho úhlem umístění

Obrázek 3.2: *Hmatník, podle kterého se mapuje matice homografie, model Epiphone SG 400 (menzura 630mm)*

V rámci knihovny OpenCV existuje funkce `cvFindHomography()` [13], která pro tento projekt sloužila jako kontrolní. Umožňuje najít perspektivní transformaci homografie `H = ||hij||`, a to za použití tří metod – základní nejmenší medián a robustní RANSAC. Pro naše využití bude stačit základní implementace a v rámci této funkce je tak možné se vlastní implementací inspirovat. Další užitečná funkce knihovny OpenCV `cvWarpPerspective()` transformuje vstupní obrázek na odpovídající perspektivní transformaci, tj. aplikuje homografii na zdrojový obrázek (slouží pro kontrolu, zobrazen na obrázku 3.2).



Obrázek 3.3: Znázornění zobrazení homografie

3.2.1 Výpočet matice pro homografii

Budeme potřebovat transformovat souřadnice obrazu kamery na souřadnice pomocného obrázku hmatníku. Jak tedy získáme matici homografie *H*? Předpokladem pro získání takovéto matice je znalost alespoň 4 korespondujících bodů z obou rovin, při podmínce, že žádné tři z těchto bodů nesmí ležet v jedné přímce. Tento problém není v našem případě třeba nikterak řešit, neboť jako orientační body hmatníku byly vybrány jeho hrany (tj. obdélníkový tvar).

Bod ve snímaném obraze reprezentujeme vektorem p_a, bod na obrázku hmatníku vektorem p_b, matice homografie *H* je matice o velikosti 3x3, jak je popsáno ve vzorcích 3 a 4:

$$p_a = \begin{bmatrix} X \\ Y \\ 1 \end{bmatrix}, p_b = \begin{bmatrix} wx \\ wy \\ w \end{bmatrix}, H = \begin{bmatrix} h_{11}h_{12}h_{13} \\ h_{21}h_{22}h_{23} \\ h_{31}h_{32}h_{33} \end{bmatrix} \qquad [3]$$

$$p_b = Hp_a \qquad [4]$$

Čtyři známé body jsou pro náš případ naprosto vyhovující, nicméně zde je ukázka matice homografie pro obecný počet bodů N ve vzorci 5:

$$\begin{bmatrix} -X_1 & -Y_1 & -1 & 0 & 0 & 0 & x_1X_1 & x_1Y_1 & x_1 \\ 0 & 0 & 0 & -X_1 & -Y_1 & -1 & y_1Y_1 & y_1Y_1 & y_1 \\ -X_2 & -Y_2 & -1 & 0 & 0 & 0 & x_2X_2 & x_2Y_2 & x_2 \\ 0 & 0 & 0 & -X_2 & -Y_2 & -1 & y_2Y_2 & y_2Y_2 & y_2 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ -X_N & -Y_N & -1 & 0 & 0 & 0 & x_NX_N & x_NY_N & x_N \\ 0 & 0 & 0 & -X_N & -Y_N & -1 & y_NY_N & y_NY_N & y_N \end{bmatrix} \qquad [5]$$

Pro úpravu matice využijeme principů Gaussova eliminační metody. Soustavu lineárních algebraických rovnic lze vyjádřit rozšířenou maticí soustavy. Řádkovými (nikoliv sloupcovými) úpravami převádíme tuto matici do tvaru, kdy se pod hlavní diagonálou nachází pouze nuly. Upravená matice pak odpovídá soustavě rovnic, která je ekvivalentní s původní soustavou [zdroj]. Matici tedy upravíme na trojúhelníkový tvar a odpovídající body pak použijeme pro sestrojení samotné 3x3 matice homografie.

3.3 Princip rozřazení pražců

Původně bylo uvažováno nalezení jednotlivých pražců pomocí detekční hranové funkce (konkrétně horizontálního Sobelova operátoru). Nicméně tato funkce je (testováno na nezanedbatelné škále parametrů) značně nedostačující pro většinu testovaných prostředí. Proto jsem se rozhodl implementovat rozřazení pražců čistě podle geometrických proporcí dnešních typů kytar a obecné rovnice pro výpočet rozsahu pražců podle menzury, viz vzorec 6:

$$d = s - \left(\frac{s}{2^{\frac{n}{12}}} \right) \qquad [6]$$

Uživatel si v možnostech nastavení vybere, kolikapražcový jeho nástroj je a podle délky nalezeného hmatníku se ukládají rozsahy jednotlivých pražců do struktury. Tento způsob značně zjednodušuje robustnost výpočetních operací a program pracuje se zachováním přibližně stejné úspěšnosti výsledků jako v případě výše zmíněného způsobu. Tabulka 1 popisuje uvažované typy nástrojů:

elektrické, jazzové kytary (Gibson)	630 mm (24,75")
elektrické kytary (PRS)	635 mm (25")
elektrické, jazzové kytary, akustické kytary (Fender)	650 mm (25,5")
klasické kytary	660 mm (26")
gypsy kytary	670 mm (26,25")

Tabulka 1: Uvažované typy nástrojů

4 Implementační prvky

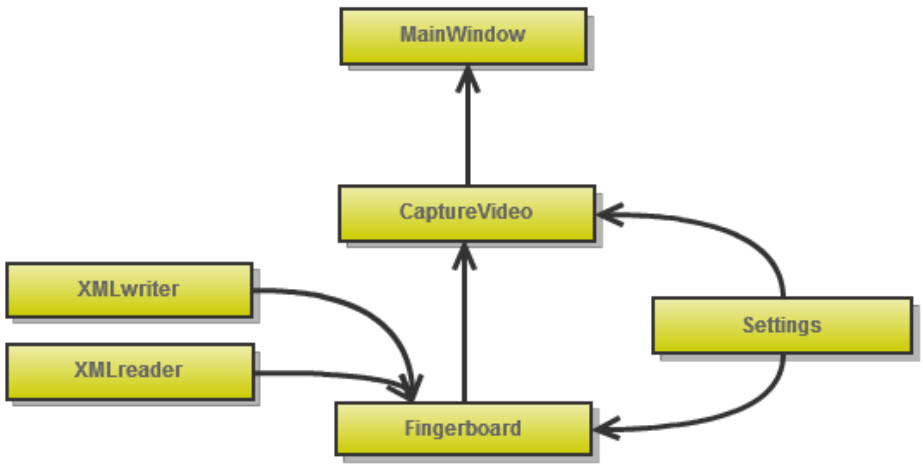
Tato kapitola popisuje zásadní části implementační části aplikace, dále popisuje prostředky a některé problémy vznikající právě při implementaci.

4.1 Objektový návrh aplikace

Celý projekt je zpracován v jazyce C++ a dodržuje požadavky na správný objektový návrh. V následujících podkapitolách si rozebereme třídní hierarchii a dále vlastnosti jednotlivých tříd. Rovněž zde zmíním implementační informace ohledně rozdělení nástroje na pražce a formát uložení daného trénovaného prvku (akordu/stupnice).

4.1.1 Hierarchie tříd

Na obrázku 4.1 je znázorněna kooperace jednotlivých tříd programu. Největší váhu má třída zobrazena dole, tzn. například třídě `Fingerboard` jsou závislé všechny výše označené a šipkou spojené třídy. V rámci návrhu tříd z hlediska hierarchie jsem se snažil o co největší kompaktnost, jednoduchost a smysluplnost v případě dalších úprav či budování rozšíření. V následujícím textu si jednotlivé třídy popíšeme a zpřesníme jejich účel v rámci projektu.



Obrázek 4.1: Hierarchie tříd programu

4.1.2 Třída Settings

Třída Settings slouží pro uchování veškerých hodnot nastavení programu. Právě v této třídě se zpracovávají uživatelem zvolené parametry konkrétního nástroje, barva identifikačních značek a tolerance těchto barev. Z hlediska implementace se nejedná o třídu, která by byla v některém ze svých prvků kódu zajímavá. Metoda pro výběr barvy `RGBChange(int)` je definována následujícím kódem:

```
void Settings::RGBChange(int currentIndex) {
    switch(currentIndex) {
        case CB_RED:
            lastColor = QColor(255, 0, 0);
            break;
        case CB_GREEN:
            lastColor = QColor(0, 255, 0);
            break;
        case CB_YELLOW:
            lastColor = QColor(255, 255, 0);
            break;
    }
    QBrush brush(lastColor);
    ui->graphicsView->setBackgroundBrush(brush);
    ui->graphicsView->backgroundBrush();
}
```

V rámci implementace této třídy bylo zřejmě nejproblematičtějším úsekem řešit rozdílný přístup k datové reprezentaci barvy v knihovnách OpenCV a Qt (více viz kapitola X.X). S třídou pro nastavení rovněž souvisí možnost uchovávat si již nastavená data v konfigurační XML souboru (to zejména proto, že jeden uživatel zřejmě svůj nástroj výrazně často měnit nebude a není proto nutnost cokoliv znovu nastavovat). Konfigurační soubor, jehož načítání a zpracování řeší třídy `XMLreader` a `XMLwriter` (které není třeba nijak obšírně představovat; jedná se o prosté zpracování, respektive vytváření XML souboru), má následující strukturu:

```
<?xml version="1.0" encoding="UTF-8" ?>
<Config>
  <Guitar_Type>650</Guitar_Type>
  <Body_Color>255,0,0</Body_Color>
  <Head_Color>0,255,0</Head_Color>
  <Tolerance>13</Tolerance>
</Config>
```

Jak lze z ukázky vidět, ukládají se pouze čtyři vlastnosti nastavení programu, které jsou nicméně postačující k tomu, aby uživatel při následujících spuštěních nemusel znovu hodnoty připravovat. V důsledku jednoduchosti konfiguračního souboru není nutné řešit změnu jedné vlastnosti nějak specificky, ale pokaždé se celý soubor přepíše.

4.1.3 Třída Fingerboard

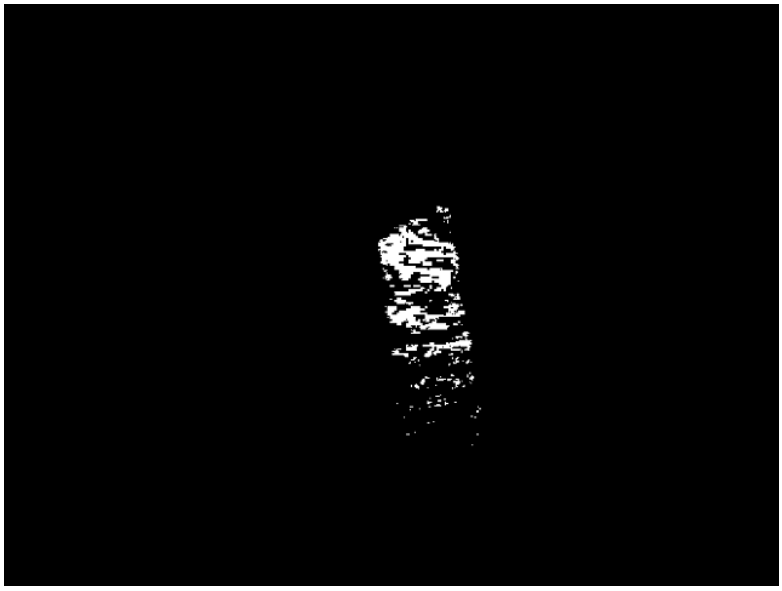
Třída `Fingerboard` popisuje veškeré náležitosti, které je potřebné zajistit v rámci detekce hmatníku. Obsahuje například funkce `countFretPosition()`, `doHomographyEstimation()`, `findVerticalBoundaries()`, jejichž název je dostatečně vypovídající. Třídní metody slouží pro kompletní detekci hmatníku, následný výpočet homografického zobrazení a transformaci jednotlivých tónů do obrazu snímaného kamerou. Ačkoliv je třída z hlediska implementace poměrně robustní, uchovává veškeré informace potřebné pro práci s hmatníkem.

Implementace detekce barevných značek

Jelikož výše bylo již několikrát zmíněno, že samotná detekce hmatníku sestává ze dvou částí, odpovídají tomuto návrhu i implementační funkce `findVerticalBoundaries()` a `findHorizontalBoundaries()`. První zmíněná řeší nalezení barevných značek v obraze. Výsledku dosahuji v několika krocích:

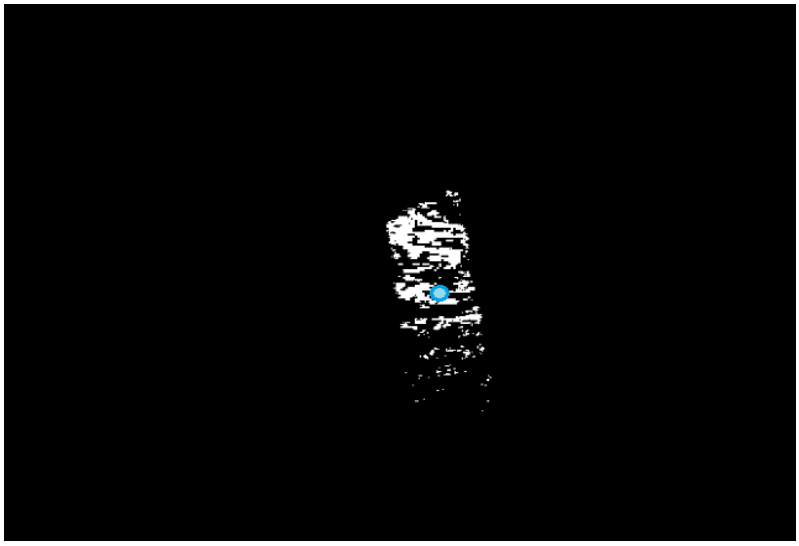
1) Výše zmíněné funkci předávám snímek, který převádím pro jednodušší orientaci na model HSV (viz kapitola 4.7.4)

2) Tento snímek je zpracován pomocí funkce `cvInRangeS()` s odpovídajícími parametry – nalezne všechny části obrazu s barvou odpovídající zadaným parametrům (např. červenou), počítá i s předem určenou tolerancí (viz obrázek 4.2)



Obrázek 4.2: Obraz po aplikaci funkce cvInRangeS()

3) Pomocí aplikace funkcí `cvMoments()` a `cvGetCentralMoment()` na obraz získám aktuální střední polohy (viz kapitola 4.7.3) daných barev (viz obrázek 4.3)



Obrázek 4.3: Obraz po získání středního momentu

4) Takto získané polohy zasílám třídě `CaptureVideo` pro další zpracování

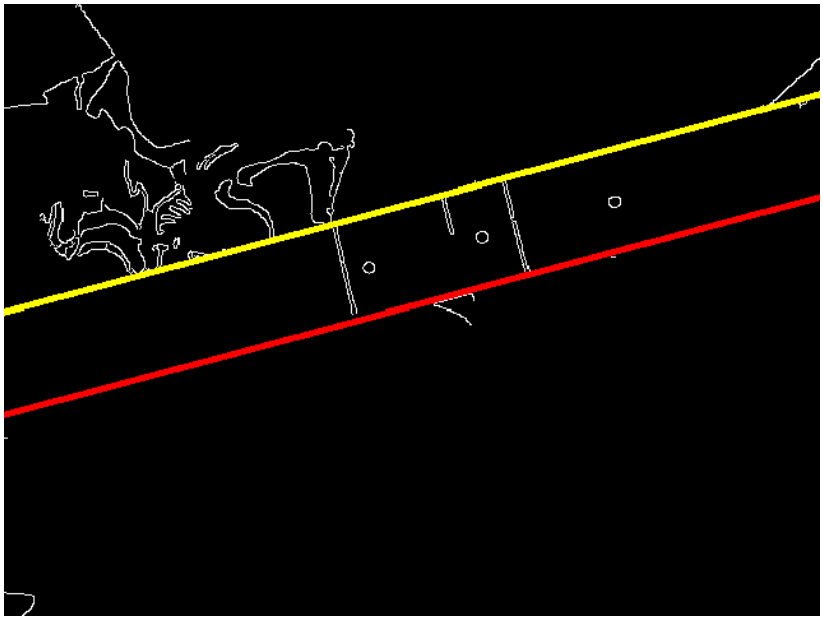
Implementace detekce horizontálních hran hmatníku

Z hlediska náročnosti implementace část identifikace horizontálních hran neméně náročnější než část předchozí. Skládá se z následujících kroků:

1) Aplikace hranové funkce `cvCanny(color_src, dst, 50, 200, 3);`

2) Aplikace Houghovy transformace (viz kapitola 3.1)

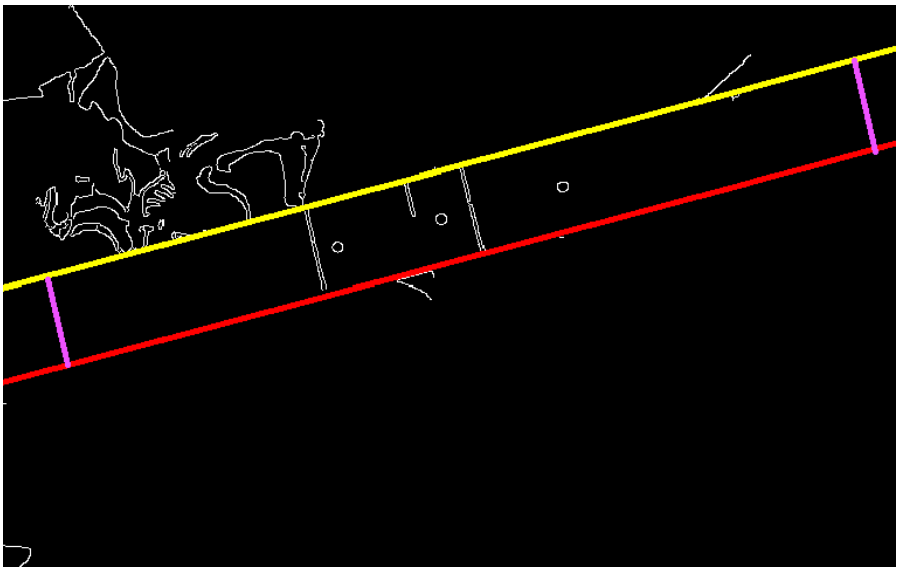
3) Zachycení deseti nejvýraznějších čar, z nichž musí být vždy právě dvě rovnoběžné, mít v rámci snímku co největší vzdálenost (nesmí být identické apod.) a nejsou vertikální (viz obrázek 4.4)



Obrázek 4.4: Obraz po nalezení horizontálních hranic hmatníku

4) Dva a dva body těchto přímek jsou odevzdány třídě `CaptureVideo`

Výsledné spojení těchto dvou funkcí (po doplnění výpočtu normálových přímek, které jsou kolmé na přímky horizontálních hran a zároveň procházejí nalezenými středy značek) graficky popisuje obrázek 4.5:



Obrázek 4.5: Obraz úspěšné detekce hmatníku

Implementace homografie

Po úspěšné detekci hmatníku míří implementace k nalezení zobrazení homografie mezi referenčním obrázkem a snímkem s definovaným hmatníkem. Výpočet je řešen vyplněním matice uvedené v kapitole 3.4.1 a následné aplikaci Gaussovy eliminace.

Koordináty rohů pomocného obrázku hmatníku jsou striktně dány jeho velikostí:

```
float src[4][2]  =   {  {0,  0},
                        {1399, 20},
                        {0,  169},
                        {1399, 152}
   };
```

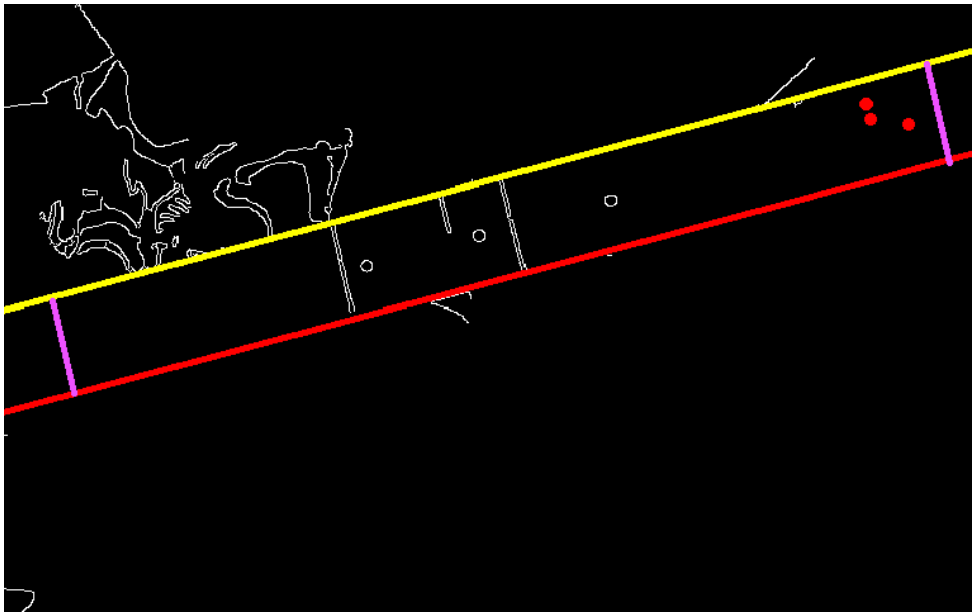
Z práce programu (detekce hmatníku) pak můžeme získat např. tyto výsledky:

```
float dst[4][2]  =   {  {262,  268},
                        {364, 381},
                        {282,  362},
                        {382, 412}
   };
```

Kdy pořadí bodů musí být pro korektní funkčnost samozřejmě dodrženo. Konkrétní matice homografie může vypadat například takto:

$$\begin{matrix} 0.0213 & -0.13854 & -0.4321 \\ 2.16567 & 1203.192 & 0.78541 \\ 1189.0358 & -0.00564 & 1 \end{matrix}$$

Homografií zpracovaný snímek s vykreslenými body pak vypadá jako na obrázku 4.6:



Obrázek 4.6: Vykreslený akord E-Dur

4.1.4 Třída CaptureVideo

Třída `CaptureVideo` slouží jako třída, která zaštiťuje vykonávání všech akcí uvedených v předchozím textu v souvislosti se získáním snímku a správou paměti. Propojuje funkčnost jednotlivých ovládacích prvků se třídou `MainWindow`, ve které z hlediska knihovny Qt funguje jako objekt `QWidget`. Příprava snímku pro aplikaci výše zmíněných prací s obrazem vypadá následovně:

```
m_i = QImage(QSize(frame->width,frame->height),QImage::Format_RGB888);
ui.frame->setMinimumSize(m_i.width(),m_i.height());
ui.frame->setMaximumSize(ui.frame->minimumSize());
m_opencvimg = cvCreateImageHeader(cvSize(m_i.width(),m_i.height()),8,3);
          m_opencvimg->imageData = (char*) m_i.bits();
```

Program připraví snímek pro formát knihovny OpenCV, protože se snímkem se dále bude pracovat jen v rámci právě této grafické knihovny. Dále tato třída řeší propojení mezi třídou `Fingerboard` a výběrem trénovacího prvku z nabídky v hlavním okně. Tzn. je potřeba podle typu trénovacího prvku upravit imaginární pozice tónů v referenčním obrázku – tato úprava vypadá například pro strunu E0 takto:

```
chordPosVer[0] = f->countFretPosition(xr->getE0ScaleAt(i).at(0).toInt(),
f->getFretboardScale());
```

V cyklu zachycení snímku pak probíhají všechny operace se snímkem. Kladu zde důraz na co největší úsporu paměti, vzhledem k tomu, že celý cyklus se opakuje zhruba ve třiceti smyčkách za sekundu. V případě špatné detekce hmatníku načítám poslední správně uložené pozice, a to zejména proto, že nepředpokládám, že by hráč se svým nástrojem při samotné hře nějak extrémně vůči snímacímu zařízení pohyboval. V rámci popisovaného cyklu rovněž probíhá kontrola všech vlastností, které má uživatel v GUI možnost vybrat (zobrazovat detekovaný hmatník: ANO/NE, zobrazovat nulté pražce ANO/NE atp.) – výhoda tohoto řešení je, že není nikterak náročné na spotřebu paměti či

výpočetního modulu procesoru a přitom probíhá v reálném čase a uživatel si tak může libovolně v jakoukoliv dobu vlastnost modifikovat a okamžitě vidět změnu.

4.2 Reprezentace akordu a stupnice

Pro vnější reprezentaci zobrazeného akordu, potažmo stupnice, jsem zvolil jednoduchou strukturu ve formátu XML:

```
<?xml version="1.0" encoding="UTF-8" ?>
<Scale>
  <Name>C Dur [stupnice]</Name>
  <Strings>
     <StringE0>X</StringE0>
     <StringA>3,5</StringA>
     <StringD>2,3,5</StringD>
     <StringG>2,4</StringG>
     <StringB>1</StringB>
     <StringE1>X</StringE1>
  </Strings>
</Scale>
```

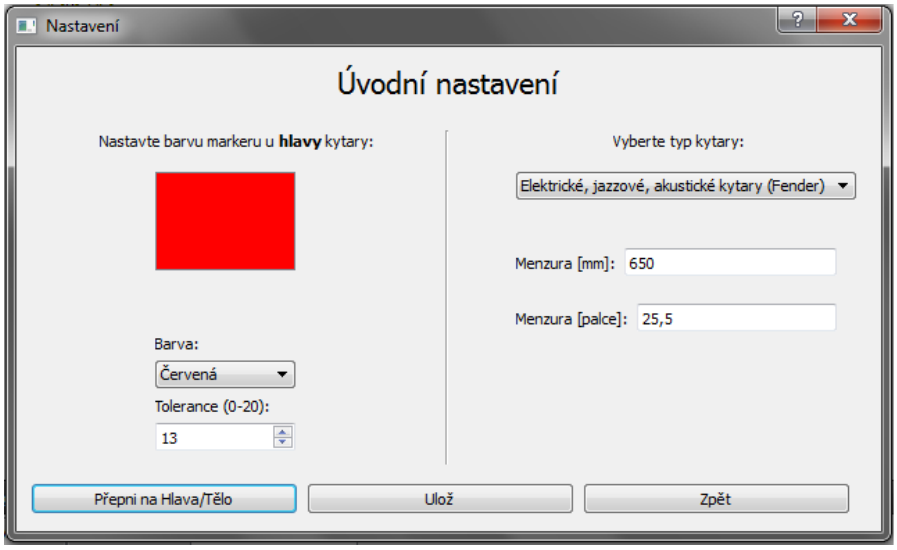
XML uchovává pouze nutné informace k identifikaci – jméno, typ (akord/stupnice) a obsažené pražce na jednotlivých strunách. To se sebou snoubí možnost rychlé úpravy i samotným uživatelem, popř. možnost vytvoření jednoduchého online editor pro akordy/stupnice méně obvyklé.

4.3 Princip mapování tónů

Jednotlivé tony daného akordu či stupnice jsou uloženy ve formátu XML, jehož struktura byla již výše popsána. Program pomocí třídy `XMLreader` data zpracuje a pro jednotlivé struny dopočítává pozici tónů s ohledem na délku krku a daný typ trénovaného prvku. V rámci třídy `CaptureVideo` pak dopočítává pozice tónů v referenčním obrázku, jež ukládá a v momentě, kdy získá korektní matici homografie, tak tyto pozice mapuje do snímaného obrazu.

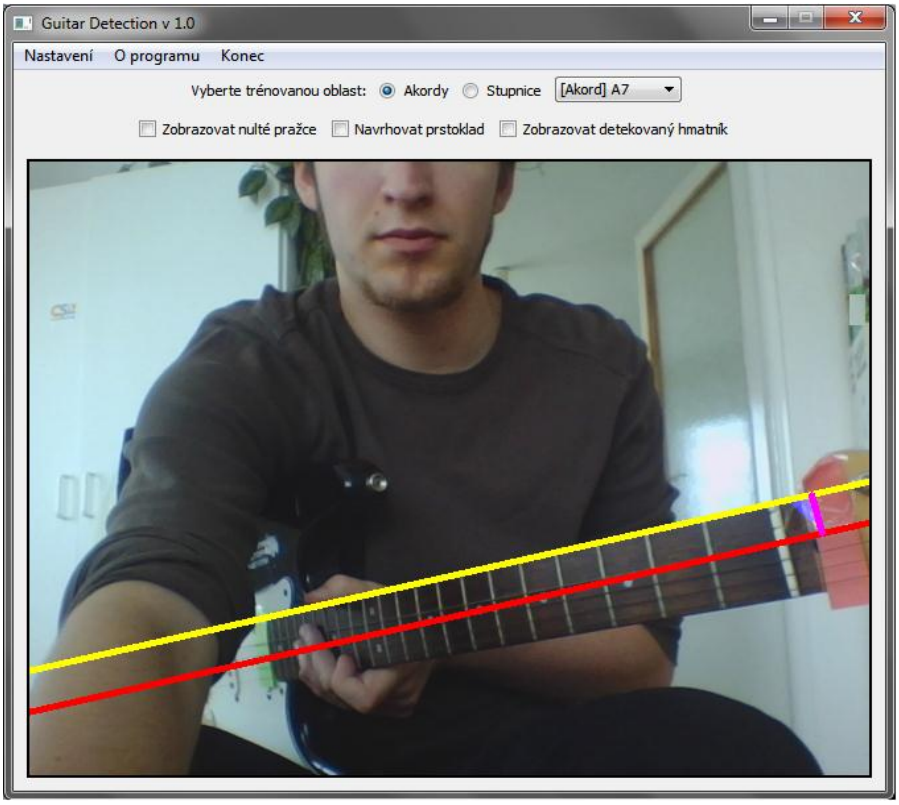
4.4 Grafické uživatelské rozhraní

Aplikace disponuje grafickým uživatelským rozhraním, které skutečně staví na tom, aby měl uživatel co nejjednodušší přístup k hlavní funkci programu, tj. viděl daný akord/stupnici přímo v obraze. Skládá se z dialogového okna pro nastavení, které popisuje obrázek 4.7:



Obrázek 4.7: Dialogové okno pro nastavení

a hlavního okna programu (viz obrázek 4.8):



Obrázek 4.8: Hlavní okno programu

4.5 Vývojové prostředí

V následující podkapitole rozebírám výhody a nevýhody použitého vývojového prostředí Qt Creator (ačkoliv jsem vyzkoušel více více vývojových prostředí, Qt Creator byl však pro potřeby projektu nejvhodnější) a uvádím bližší informace k verzovacímu systému SVN, který jsem v rámci řešení často využíval a ukázal se jako velmi užitečný prostředek k zachovávání jednotlivých verzí programu:

4.5.1 Qt Creator

Projekt byl vyvíjen v prostředí Qt Creator. Konkrétně se jedná o v dnešní době jedno z nejsilnějších otevřených IDE[1], které mnoha způsoby nepřímo dohlíží na kvalitu kódu v průběhu celého životního cyklu aplikací. Verze, se kterou jsem pracoval, je založená na nejaktuálnější verzi knihovny Qt, je zatím nejintuitivnější a jednoduchá k použití, přičemž obsahuje veškeré textové a ladící nástroje, které programátor pro psaní vyžaduje. Rovněž disponuje kvalitní kontrolou syntaktických chyb při psaní kódu.

4.5.2 SVN

SVN (zkratka pro plný název Subversion) je verzovací systém pracující na architektuře typu klient/server. Zálohovat pravidelně projekt se ukázalo jako velmi potřebné, a to nejen z důvodu možné ztráty dat. Ačkoli Subversion pracuje pouze s připojením k internetu, neukázal se tento nedostatek jako fatální. Navíc je velmi dobře provázaný s výše uvedeným vývojovým prostředím. SVN umožňuje každou verzi komentovat, což značně usnadňuje jejich zpětné prohledávání. Upravené soubory jsou ve standardním prohlížeči lehce rozpoznatelné. Rovněž je možno filtrovat pouze odlišné části zdrojového kódu.

4.6 Použité knihovny

Nejpodstatnější část implementace tvoří logické bloky funkcí využitých knihoven, které společně se stručným popisem uvádím v této podkapitole.

4.6.1 QT

QT je grafická knihovna (implementovaná v jazyce C++), která umožňuje vývojářům vytvářet aplikace pro libovolný operační systém 32-bitové nebo 64-bitové architektury. Mimo implementaci v C++ má QT slibně se vyvíjející implementace I v jazycích Python, Perl nebo Ruby. Na rozdíl od jiných platforem sady nástrojů, QT poskytuje její aplikace skutečně nativní vzhled, protože používá platformy nativního API. Předností QT je dobrá dokumentace a velmi svižné opravování problémů a reakce vývojářů na případné stížnosti programátorů.

V rámci tohoto projektu je QT využito pro zprostředkování samotného grafického rozhraní ve všech směrech.

[1] Užívaný termín pro vývojové prostředí – angl. zkratka pro „Integrated Development Environment“

4.6.2 OpenCV

OpenCV je knihovna pro zpracování obrazu. Funguje pod BSD licencí (čili je zdarma pro akademické i komerční využití). Je portována do jazyků C++, C, Python a v budoucnu i do Javy, z hlediska operačních systémů funguje pod následujícími: Windows, Linux, Mac a Android. Její možnosti využívají široké škály oborů od interaktivních umění přes obrazové webové aplikace až po pokročilou robotiku.

4.6.3 ARToolkit

ARToolKit je C (portovaná I na jazyk C++) softwarová knihovna, která umožňuje programátorům snadno vyvíjet aplikace spolupracující s rozšířenou realitou (angl. Augmented Reality). Rozšířená realita má mnoho potenciálních aplikací v průmyslu a akademického výzkumu.

ARToolkit v mojí aplikaci původně zajišťoval reálný tracking hmatníku kytary. Používá techniky počítačového vidění pro výpočet skutečné pozice a orientace kamery vzhledem ke čtyřem markerům. Díky těmto rychle a přesně získaným polohám lze extrahovat body hmatníku pro další použití s knihovnou OpenCV. Aktuální implementace však řeší tracking pouze v rámci OpenCV, přesto zde ARToolkit zveřejňuji, protože mi v mnohých aktuálně používaných postupech naznačil správnou cestu, jak k problému přistupovat.

4.7 Implementační zajímavosti

V této podkapitole uvádím několik zajímavých podnětů k dalšímu studiu, které se naskytly během implementace, a jejich pochopení značně usnadnilo celý proces vyvíjení programu. Jedná se o průřez celým projektem a jednotlivé části spolu nikterak významně nesouvisí.

4.7.1 Správa paměti v OpenCV

Jistým problémem při implementaci funkcí pracující s knihovnou OpenCV je nutnost neustále hledět na složku rychlosti zachytávaného videa. Jelikož se ve smyčce pro práci s aktuálním snímkem vykonává spousta složitých matematických operací, je potřeba správně a především včas uvolňovat nepoužívanou paměť. V případě neúplného dohledu nad množstvím aktuálně probíhajících operací se mi několikrát přihodilo, že program běžel ve standardním rozlišení 640x480px například na úrovni pouhých 10 FPS[2]. Pro korektní správu paměti v OpenCV existuje několik dobrých materiálů [1]. Po včasné dealokaci nepoužívaných komponent, častějšímu využívání třídy `cvMemStorage` namísto neumělého `cvArr` nebo `cvSeq` a vůbec další aplikované snaze o čistší kód, se problémy s nízkým FPS vyřešily.

[2] počet snímku za sekundu – angl. zkratka pro „Frame Per Second“

4.7.2 Převod formátu QImage na IplImage a naopak

Jelikož tento projekt z hlediska implementace koordinuje funkčnost dvou knihoven (OpenCV a Qt), z nichž má každá jiný přístup k datové reprezentaci obrazu, bylo nutné tento problém řešit napsáním vlastní konverzní funkce. V podstatě se jedná o pouhé naplnění jednotlivých položek struktury odpovídajícími položkami struktury druhé. Nicméně zjištění, jak odlišně tyto knihovny s obrazem na datové bázi pracují, mi značně ulehčilo fázi ladění programu. Následuje ukázka té jednodušší funkce na převod – v tomto případě `QImage` na formát OpenCV `IplImage`:

```
IplImage * qImg2IplImg(QImage *qi) {
	IplImage * imgHeader = cvCreateImageHeader( cvSize(qi->width(), qi-
	>height()), IPL_DEPTH_8U, 4);
	imgHeader->imageData = (char *) qi->bits();
	uchar* newdata = (uchar *) malloc(sizeof(uchar) * qi->byteCount());
	memcpy(newdata, qi->bits(), qi->byteCount());
	imgHeader->imageData = (char*) newdata;
	return imgHeader;
}
```

4.7.3 Funkce pro získání středního momentu

Knihovna OpenCV nabízí pro práci s momenty, jako fyzikální veličinou řadu užitečných funkcí. V rámci tohoto projektu jsem pro nalezení centrálního bodu barevné značky využil implementace funkce `getCentralMoment()`. Původně jen z čisté zvědavosti jsem se začetl do popisu implementace této funkce a nakonec mě problematika získávání této veličiny velmi zaujala. Ve výsledku se centrální moment daného objektu v obraze získává ze vzorce 7:

$$M_{x,y_{order}} = \sum_{x,y} I(x,y) * (x - x_c)^{x_{order}} * (y - y_c)^{y_{order}} \qquad [7]$$

Vzorec 4.1: Výpočet středního momentu ze získané intenzity všech pixelů obrazu

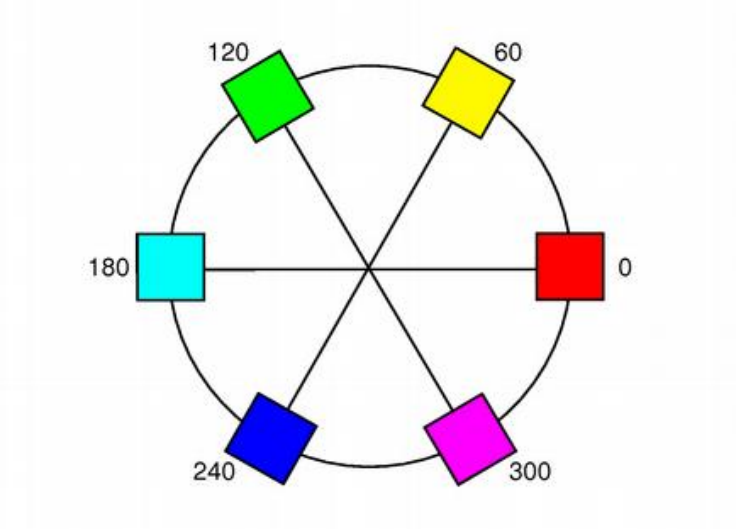
kde `I(x,y)` je intenzita aktuálně zpracovávaného pixelu se souřadnicemi (x,y).

4.7.4 Reprezentace barvy v OpenCV a Qt

V rámci projektové koexistence knihovny OpenCV s knihovnou Qt se vyskytly některé zajímavé problémy s reprezentací barvy v těchto dvou knihovnách. Zatímco OpenCV ve výchozím režimu využívá již tak dost nestandardní BGR (blue, green, red) model barvy, Qt jej modeluje v modernějším formátu HSV. BGR model má pouze tři klasické barevné složky uložené v jiném pořadí – na toto je při programování potřeba dávat dobrý pozor. Zajímavější je ale převod právě mezi tímto modelem a modelem HSV.

HSV model má stejně jako RGB model tři komponenty:

- odstín (angl. hue) – hodnoty v intervalu <0;359>, které reprezentují míru „šedivosti“, například červená barva má hodnotu odstínu rovnu nule, žlutá šedesáti, zelená stodvaceti (dále viz obrázek 4.8)
- sytost (angl. saturation) – hodnoty v intervalu <0;255>, čím je hodnota větší, tím „silnější“ je daná barva. Šedivé typy barev mají tuto hodnotu blízkou nule a například jasná červená má tuto hodnotu téměř na 255
- jas (angl. value) – hodnoty v intervalu <0;255>, nízký jas mají barvy blízké černé, vysoký naopak barvy blízké bílé



Obrázek 4.9: Grafické znázornění reprezentace sytosti (angl. hue) v knihovně Qt

5 Testování

Výsledný program prošel fází testování, která sice nezaručuje, že výsledek bude korektně fungovat se všemi druhy snímacích zařízení a prostředí stejně jako v mém případě, nicméně tuto šanci vzhledem k pestrosti testování značně navyšuje. Program byl testován na notebooku HP ProBook 4530s, který disponuje dvoujádrovým procesorem Intel Core i3 2310M s frekvencí 2,1 GHz a 4 GB DDR3 operační paměti.

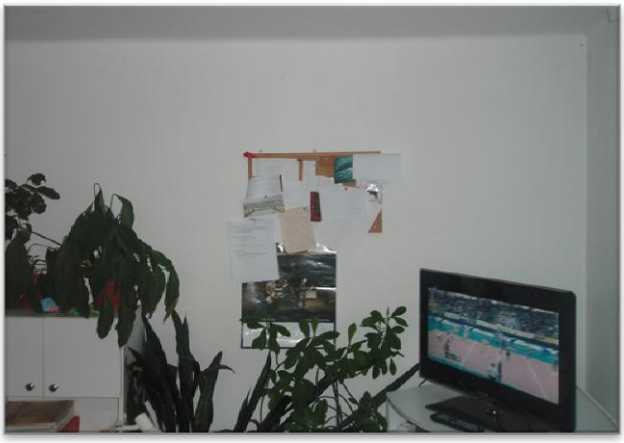
5.1 Snímací zařízení

Program byl testován na dvou různých zařízeních: integrované webkameře HD Webcam v notebooku HP ProBook 4530s a na přenosné webkameře Logitech Webcam C500. Přenosné snímací zařízení poskytovalo z hlediska kvality obrazu téměř totožné výsledky jako webkamera integrovaná, a to i po delší době testování. Jelikož jsou kamery kvalitativně na zcela jiné úrovni (Logitech: 30 snímků za sekundu, rozlišení 1280x760, wide-screen efekt, 1.3 megapixelu; integrovaná webkamera: 30 snímků za sekundu, rozlišení 640x480, 1.0 megapixelu) a přitom poskytovaly s ohledem na úspěšnost zobrazení tónů a detekce hmatníku přibližně stejné výsledky, dovoluji si usuzovat, že většina dnes dostupných modelů webkamer při normální konfiguraci bude v mezích použitelnosti fungovat.

5.2 Testovací prostředí

Jelikož bylo již od začátku návrhu jasné, že i v dnešní době velmi kvalitních snímacích zařízení pořiditelných za lidovou cenu, bude úspěšnost správné práce programu do nezanedbatelné míry záviset na „pozadí“ obrazu, na kterém je hráč snímán, bylo třeba provést v rámci testování důkladnější analýzu takových prostředí. Program byl v rámci ověřování kvality napsaného algoritmu pro detekci hmatníku kytary testován ve třech prostředích (místnostech) s různě složitými parametry morfologie a viditelnosti (viz obrázek 5.1):

- bílá stěna (pokoj č. 1) – čistá bílá stěna, židle, stůl
- plakátový pokoj (pokoj č. 2) – složitější barevné konstrukce na stěnách
- obývací pokoj (pokoj č. 3) – špatná dostupnost světla, tmavá zákoutí



a. *Testované prostředí č. 1*

b. *Testované prostředí č. 2*

c. *Testované prostředí č. 3*

Obrázek 5.1: Testovaná prostředí

Tabulka 2 popisuje závislost úspěšnosti detekce hmatníku na morfologii prostředí za hráčem:

Místo	Stupeň morfologie	Den	Večer	Počet měření	Vzdálenost od kamery	Počet úspěšných do 3 s	Počet úspěšných do 10 s	Počet neúspěšných	Koeficient úspěšnosti
Pokoj č. 1	1	ANO	NE	10	cca 80cm	9	1	0	1.0
Pokoj č. 1	1	NE	ANO	10	cca 60cm	6	2	2	0.8
Pokoj č. 2	2	ANO	NE	10	cca 60cm	7	3	0	1.0
Pokoj č. 2	2	NE	ANO	10	cca 60cm	4	4	2	0.8
Pokoj č. 3	3	ANO	NE	10	cca 60cm	4	3	4	0.7
Pokoj č. 3	3	NE	ANO	10	cca 60cm	3	3	4	0.6

Tabulka č. 2:Testování úspěšnosti detekce hmatníku v závislosti na prostředí

Z výsledků je patrné, že pro nejrychlejší úspěšnou detekci hmatníku se stává nejlepším co nejjednodušší prostředí, a to především z hlediska viditelnosti a dostupnosti světla. Při správně vybraných barvách značek (markerů) na nástroji nemá program problém tyto rozpoznat, jakkoliv je pozadí barevně rozmanité. Optimálnější pro správný běh programu bylo jednoznačně denní světlo. „Žluté“ světlo ze zářivek a žárovek nepřinášelo takovou úspěšnost výsledků. Výsledky testování rovněž ukazují, že v téměř 40 % případů je třeba na korektní detekci hmatníku několik sekund „počkat“.

6 Závěr

Na začátku této práce jsem uvedl myšlenku vytvořit nevšední učební pomůcku v oblasti výuky hry na strunný nástroj. V jednotlivých kapitolách jsem rozebral z různých úhlů aspekty vývoje takovéto pomůcky, a to jak z hlediska návrhového, tak i implementačního. První bod zadání, tedy seznámení a rozebrání principů rozšířené reality a zobrazení homografie, je popsáno v kapitolách 2 a 3. Návrh a jednotlivé aspekty týkající se především teoretických informací nutných k úspěšné implementaci rozebírá kapitola 3. Implementační řešení, které zeširoka popisuje stavbu programu, vysvětluje kapitola 4.

Dovoluji si říci, že stanovené cíle byly ve velmi přijatelné míře dosaženy. Ačkoliv jsem se po čas návrhu i implementace potýkal s různými problémy (viz kapitola 4.7) podařilo se mi je především díky základně slušných (ač v převážné míře zahraničních) materiálů vyřešit. Z hlediska teoretického se především v případě pochopení homografie a principů prostorových transformací nejedná o příliš triviální problematiku, proto se ve výsledku velmi vyplatila elektronická korespondence s několika lidmi, kteří se v oblasti matematiky a geometrie profesně pohybují.

V rámci návrhu a implementace programu jsem dostal několik nápadů, které by byly vhodné k zapracování do aplikace, ale ať již z důvodu časového nebo důvodu přílišné náročnosti přípravy a nasazení vlastnosti do projektu, se tyto do finální verze programu nedostaly. Věřím, že v budoucnu by bylo zajímavé tyto nápady do programu zapracovat.

Primární vlastností, která by si zasloužila do projektu zapracovat, je potvrzování o stisku daného tónu programem. Nicméně toto rozšíření implikuje napsání algoritmu pro detekci konečků prstů, což by samo o sobě bylo tak rozsáhlým a samostatným projektem. Nicméně by taková vlastnost značně zvýšila interaktivitu s uživatelem, který by byl v jistém slova smyslu programem hodnocen a z pozitivních výsledků by cítil motivaci k dalšímu učení.

Jelikož je formát XML souboru, který uchovává daný trénovací prvek (akord/stupnici) definován mojí osobou a není nijak normalizován, bylo by vhodné navrhnout rychlý editor, kde by si hráč jednoduše nové akordy vytvářel a systém zpětně ověřoval, jestli skutečně existují (např. náhledem do internetové databáze).

Pokud by měl hráč možnost přehrát si celou svou oblíbenou skladbu, bylo by to pro něj patrně vrcholem toho, co od programu očekával. Tato vlastnost ale uvažuje implementaci časovače, který by v souvislosti se změnou akordu ve skladbě měnil i zobrazované tóny. Již samotný návrh takovéto funkčnosti je pro poměry této práce příliš robustní, nicméně z hlediska zábavy by se jednalo o velmi zajímavou vlastnost.

Celý systém je otestován za různých morfologických a světelných podmínek. Systém byl testován s dvěma odlišnými webkamerami a zaručuje korektnost fungování na většině dnešních standardizovaných přístrojů.

Věřím, že mnou navržený systém může být přínosem pro další studium a práci v rámci této specifické problematiky.

Literatura

[1] BRADSKI, G. R.; KAEHLER, A.: *Learning OpenCV: Computer Vision with OpenCV library*, 2nd edition, O'Reilly Media, Sebastopol: 2008, ISBN: 9780596516130

[2] PAYA, D. F.: *Object detection in low resolution video sequences*, Florida Atlantic University, Boca Raton: 2009

[3] Ronald AYUMA, R.; BAILLOT, Y.: *Recent Advances In Augmented Reality*, HRL Laboratiories, 2001

Elektronické zdroje

[4] BANERJEE, S.: How to compute a homography. *Computer Vision* [online]. 2008, 2008-01-20 [cit. 2012-01-12]. Dostupné z: http://www.cse.iitd.ac.in/~suban/vision/geometry/node24.html

[5] ESTRADA, J.; JEPSON, A.D.; FLEET, D.: Planar Homographies. [online]. 2004, 2004-04-15 [cit. 2012-01-12]. Dostupné z: http://www.cs.utoronto.ca/~strider/vis-notes/tutHomography04.pdf

[6] *Hledání parametrického popisu objektů pomocí Houghovy transformace*. [online]. 2008, 10/24/2008 [cit. 2012-01-12]. Dostupné z: http://cmp.felk.cvut.cz/cmp/courses/ZS1/Cviceni/cv5/hough.htm

[7] ROTH, G.: Homography [online]. 2002-02-12 [cit. 2012-04-23]. Dostupné z: http://people.scs.carleton.ca/~c_shu/Courses/comp4900d/notes/homography.pdf

[8] QUESTED et al.: Polyphonic Note Tracking Using Multimodal Retrieval of Musical Events [online], University of Leeds, UK, 2007, [cit. 2012-04-23]. Dostupné z: http://www.comp.leeds.ac.uk/roger/Research/Publications/Garry08.pdf

[9] SINHA, U.: Tracking colored objects in OpenCV. *AI Shack* [online]. 2010, 2010-07-19 [cit. 2012-01-12]. Dostupné z: http://www.aishack.in/2010/07/tracking-colored-objects-in-opencv/

[10] MILLER, K.: Guitar Hero & Rock Band Survey. [online]. 2009, [cit. 2012-04-23]. Dostupné z: http://www.brown.edu/Project/Music/guitarherosurvey.html

[11] CICCONET, M.: Edge Methods for Guitar ROI Detection. [online]. 2009, [cit. 2012-05-01]. Dostupné z: http://w3.impa.br/~cicconet/thesis/guitar_roi_detection/index.html

[12] CICCONET, M.: Infrared Methods for Guitar ROI Detection. [online]. 2009, [cit. 2012-05-01]. Dostupné z: http://w3.impa.br/~cicconet/thesis/ir_guitar_roi_detection/index.html

[13] Camera calibration and 3D reconstruction. *OpenCV Reference 2.0* [online]. Dostupné z: http://opencv.willowgarage.com/documentation/camera_calibration_and_3d_reconstruction.html

Seznam obrázků

2.1 Ukázka požadovaného umístění značek na kytaře. 4

3.1 Popis hledané přímky (parametrický přístup) 8

3.2 Hmatník, podle kterého se mapuje matice homografie. 10

3.3 Znázornění zobrazení homografie. 10

4.1 Hierarchie tříd programu. 13

4.2 Obraz po aplikaci funkce cvInRangeS(). 15

4.3 Obraz po získání středního momentu. 15

4.4 Obraz po nalezení horizontálních hranic hmatníku. 16

4.5 Obraz úspěšné detekce hmatníku. 17

4.6 Vykreslený akord E-Dur. 18

4.7 Dialogové okno pro nastavení. 20

4.8 Hlavní okno programu. 20

4.9 Grafické znázornění reprezentace sytosti v knihovně Qt. 24

5.1 Testovaná prostředí. 26

Dodatek A

Obsah CD

- Složka bin:
  - Zkompilovaná aplikace pro operační system MS Windows (spustitelný soubor)
- Složka doc:
  - Dokumentace ve formátu PDF
- Složka media:
  - Video s předvedením funkčnosti aplikace
- Složka src:
  - Zdrojové kódy aplikace
  - Složka files: soubory potřebné pro běh aplikace (tyto se nakopírují k spustitelnému programu)