VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

DETEKTOR POHYBU PRO PLATFORMU ANDROID

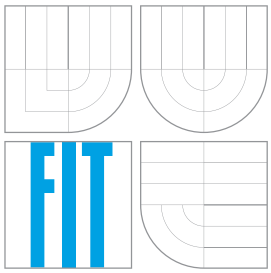
BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR

MICHAL KOMÁR

BRNO 2011

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

DETEKTOR POHYBU PRO PLATFORMU ANDROID

MOTION DETECTOR FOR ANDROID PLATFORM

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — MICHAL KOMÁR
AUTHOR

VEDOUCÍ PRÁCE — Ing. ALEŠ LÁNÍK
SUPERVISOR

BRNO 2011

Abstrakt

Tato bakalářská práce se zabývá implementací detektoru pohybu pro mobilní telefony s operačním systémem Android. Aplikace funguje jako detektor pohybu, se schopností upozornit na pohyb zaznamenaný vestavěným fotoaparátem. Jádro detektoru je napsáno v nativním kódu pro rychlejší zpracování na procesorech architektury ARM. Aplikace umožňuje různá nastavení pro optimalizaci detektoru v rozličných prostředích.

Abstract

This bachelor´s thesis deals with implementation of the motion detector for Android platform mobile phones. It works as a motion detector, with the ability to indicate moving object recorded by an integrated camera. The detector core is written in native code for faster processing by ARM architecture processors. The application allows various settings to optimize the detector in different environments.

Klíčová slova

Android, OpenCV, Cortex-A8, ARMv7, NEON, počítačové vidění, detektor pohybu, mobilní telefony.

Keywords

Android, OpenCV, Cortex-A8, ARMv7, NEON, computer vision, motion detector, mobile phones, mobile applications.

Citace

Michal Komár: Detektor pohybu pro platformu Android, bakalářská práce, Brno, FIT VUT v Brně, 2011

Detektor pohybu pro platformu Android

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením Ing. Aleše Láníka Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Michal Komár
28. dubna 2011

Poděkování

Na tomto místě bych rád poděkoval Ing. Aleši Láníkovi, vedoucímu bakalářské práce, za odbornou pomoc, ochotu a čas, který mi při tvorbě této práce věnoval.

Obsah

Kapitola 1

Úvod

Oblast mobilních zařízení je velmi dynamickou a rychle se rozvíjející. Programy, které byly dříve výsadou výkonných domácích počítačů, jsou v dnešní době úspěšně portovány pro různá mobilní zařízení a lidé tak s sebou mohou v kufříku či kapse nosit spoustu užitečného softwaru. Je tomu tak díky vyspělé technologii využívané u přenosných zařízení. Gigahertzové procesory již nejsou u mobilních telefonů výjimkou a objevují se i telefony s procesory vícejádrovými.

Nárůst výkonu, cenová dostupnost a možnost využití periferií, které nejsou u osobních počítačů běžné či praktické, jako třeba fotoaparát, GPS modul, kompas, akcelerometr a další, otevírají vrátka vývoji zcela nových aplikací.

Zajímavou problematikou, která je u mobilních telefonů teprve v plenkách, je zpracování obrazu. Právě na tuto oblast jsem se zaměřil ve své bakalářské práci a s využitím platformy Android, jejíž součástí je moderní operační systém pro mobilní telefony a jiná přenosná zařízení, bych rád demonstroval její schopnosti. Jako součást bakalářské práce jsem vytvořil aplikaci sloužící k detekci pohybu ve video sekvenci. Aplikace je zaměřena na maximální využití cílové platformy (využití nativního kódu pro náročné operace s obrazem) a dostupného hardwarového vybavení (s ohledem na moderní procesory).

Kapitola 2 této práce je zaměřena na teorii. Obsahuje úvod do technologie Android a popisuje některé její klíčové prvky. V této kapitole je také krátké seznámení s knihovnou OpenCV a teoretický rozbor pojmů a algoritmů, použitelných pro detekci pohybu, která je založena na odečítání pozadí. V kapitole 3 je popsána vlastní implementace problému a vyhodnocení výsledků vytvořených testů na různých mobilních telefonech s různou konfigurací. Je zde porovnán výkon aplikace v závislosti na hardwarových prostředcích.

Kapitola 2

Úvod do problematiky

2.1 Android

Platforma Android je softwarový balíček určený pro malá přenosná zařízení, jako jsou mobilní telefony, PDA, GPS navigace, notebooky a tablety. Android obsahuje operační systém a klíčové aplikace využívající potenciálu těchto zařízení.

2.1.1 Historie

Na počátku vývoje stála malá společnost s názvem Android Inc., založená čtveřicí vývojářů Andy Rubinem, Richem Minerem, Chrisem Whitem a Nickem Searsem. V roce 2005 byla tato firma odkoupena společností Google a Andy Rubin byl jmenován vedoucím projektu, jehož cílem bylo vytvořit pružnou a snadno aktualizovatelnou platformu pro mobilní telefony - Android. V roce 2007 Google inicioval vytvoření skupiny OHA (Open Handset Alliance), s cílem vytvářet otevřené standardy pro mobilní zařízení. Skupinu tvořilo 34 hlavních členů z řad mobilních operátorů, výrobců mobilních zařízení, softwarových vývojářů a dalších (jmenovitě například společnosti NVIDIA, Intel, Motorola, LG, T-Mobile, China Telecom, atd.). Založení OHA vyústilo v odvetu ze strany konkurence (Microsoft, Apple, Symbian). Na trh byly vypuštěny Windows Mobile 6.0 od Microsoftu, Symbian OSv9.5 a Apple iPhone. Odpovědí ze strany OHA byl Google Android, první skutečně otevřená platforma pro mobilní telefony, běžící na Linuxovém jádře, s čistým a jednoduchým uživatelským rozhraním a aplikacemi psanými v programovacím jazyce Java. Místo toho, aby bylo vytvořeno jediné konkurenční zařízení, vzniklo tak prostředí, které bylo možno integrovat do značného množství již existujících telefonů a které zároveň poskytlo výrobcům mobilních zařízení a mobilním operátorům významnou svobodu, volnost a flexibilitu při navrhování nových produktů.[10]

Verze Androidu

V současnosti je nejnovější verzí Androidu verze **3.0** s označením **Honeycomb**, která je optimalizována pro zařízení s velkými displeji. Mimo jiné nabízí také třídimenzionální plochu a upravený multi-tasking. Této verzi však předcházelo mnoho jiných [1]. Ve stručnosti:

- **0.9**

  verze vydaná v srpnu roku 2008. Obsahuje aktualizované a rozšířené API a zdokonalené vývojové nástroje.

- **1.0**

  ze září 2008. Tato verze především opravuje chyby té předchozí.

- **1.1**

  dostupná od března 2009. Významné změny zaznamenalo uživatelské rozhranní, byla přidána podpora vyhledávání hlasem. Došlo k úpravě některých vestavěných aplikací. Zařízení pro vývojáře mohla začít pracovat s komerčními aplikacemi.

- **1.5 (Cupcake)**

  verze z května 2009, podporuje nahrávání videa, stereo Bluetooth profily, nová verze klávesnice s predikcí psaného textu.

- **1.6 (Donut)**

  ze září 2009. Verze nabízí vylepšení Android Marketu, podporu pro WVGA displeje, "text-to-speech engine"pro strojové čtení textu, indikátor stavu baterie a zrychlení vyhledávání.

- **2.0/2.1 (Eclaire)**

  z konce října 2009. Novinkami verze jsou optimalizovaná rychlost hardwaru, podpora pro různá rozlišení a velikosti displejů, úprava uživatelského rozhranní, změna Google map, přidána je podpora Microsoft Exchange.

- **2.2 (Froyo)**

  umožňuje instalovat aplikace na paměťovou kartu, vytvářet wi-fi hotspot, více nastavení fotoaparátu/kamery, přibyla podpora OpenGL ES 2.0 a díky JIT (just-in-time) kompilátoru byl zvýšen výkon systému až několikanásobně.

- **2.3 (Gingerbread)**

  přináší další zdokonalení uživatelského rozhraní, lepší správu prostředků, podporu WebM přehrávání a podporu NFC (Near Field Communication), podporu více kamer a senzorů, rozšíření podpory nativního kódu a další.

Očekávaná verze ponese název **Ice Cream Sandwich**. Více informací o ní není v době tvorby této práce známo.

2.1.2 Přehled technologie

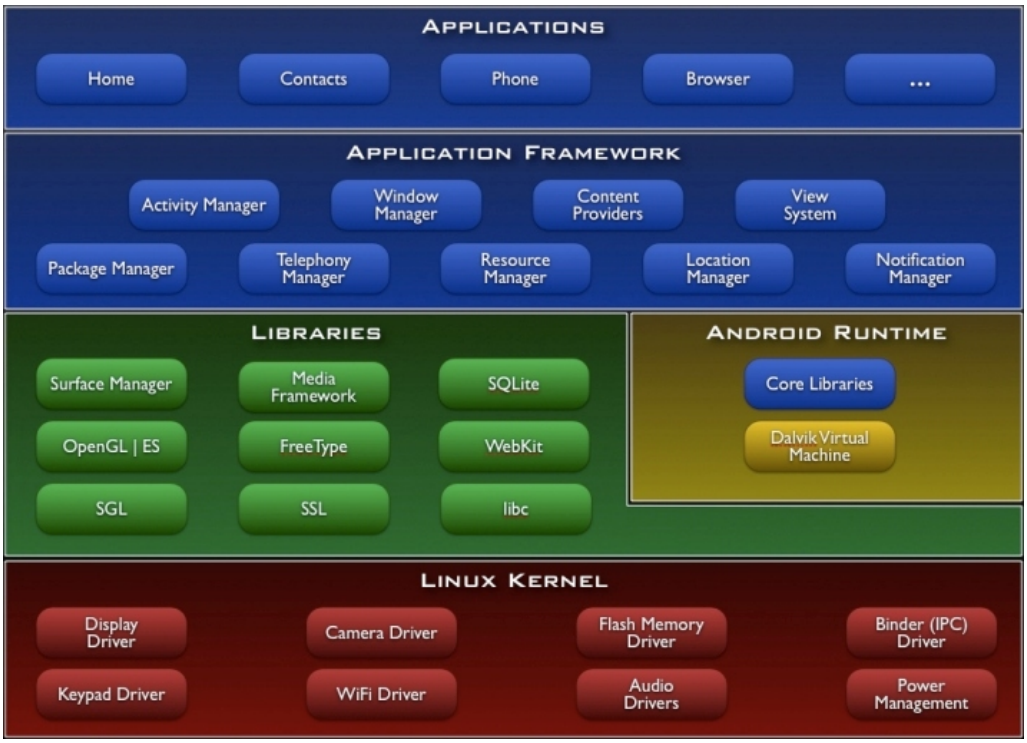
Tato podkapitola je zaměřena na klíčové části platformy Android, její architekturu, základní prvky aplikací a je zde také popsána možnost vývoje v nativním kódu. Informace jsem čerpal z oficiálních stránek podpory vývojářů pro tuto platformu [1] a z [15].

Architektura

Následující diagram 2.1 ukazuje hlavní komponenty operačního systému Android. Každá z jeho sekcí bude podrobněji popsána níže.

- **Aplikace (Applications)**

  Každá distribuce Android obsahuje základní balíček aplikací (*core applications*), zahrnující e-mailového klienta, program pro posílání a přijímání SMS, kalendář, mapy,



Obrázek 2.1: Architektura operačního systému Android. Převzato z [1]

internetový prohlížeč, kontakty a další. Všechny tyto aplikace jsou napsány s použitím programovacího jazyka Java.

- **Aplikační rozhraní (Application Framework)**

  Jakožto otevřená vývojová platforma nabízí Android vývojářům možnost tvorby velmi schopných a inovativních aplikací. Vývojářům je umožněno volně využívat hardwaru zařízení, informací o pozici, běhu služeb na pozadí, nastavování alarmu, přidávání upozornění do stavového řádku a mnohé další.

  Vývojáři mají plný přístup k celému frameworku používanému základními aplikacemi jádra. Architektura aplikací je navržena tak, aby umožňovala snadné znovupoužití jejich komponent. Aplikace může nabízet své schopnosti a jiná jich pak může využívat. Stejný mechanizmus pak umožňuje náhradu jednotlivých komponent uživatelem. Výhodou je pak jednodušší standardizace ovládání jednotlivých aplikací a možnost využití již jednou naprogramovaného.

- **Knihovny (Libraries)**

  Android obsahuje množinu C/C++ knihoven používaných různými komponentami systému. Tyto knihovny jsou vývojářům dostupné přes aplikační rozhraní. Patří mezi ně zejména systémová C knihovna, knihovny na podporu nahrávání a přehrávání médií, správce pro řízení zobrazení, grafické knihovny s podporou 2D i 3D, výkonná podpora relačních databází v podobě SQLite a další.

- **Běhové prostředí (Android Runtime)**

Každá aplikace na Androidu běží jako izolovaný proces s vlastní instancí virtuálního stroje **Dalvik** s možností využívat základních knihoven. Dalvik byl navržen tak, aby umožňoval úsporný běh mnoha virtuálních strojů na jednom zařízení. Dalvik zpracovává soubory v *.dex* formátu, které jsou vytvořeny nástrojem "*dx*"transformací z tříd zkompilovaných v Javě.

Virtuální stroj Dalvik využívá Linuxového jádra pro podporu základní funkcionality, jako je vyžití vláken nebo práce s pamětí.

- **Linuxové jádro (Linux Kernel)**

  Operační systém Android je postaven na Linuxovém jádře verze 2.6, které tvoří prostředníka mezi hardwarovým vybavením zařízení a softwarovým vybavením systému.

Android aplikace

Android aplikace jsou psány v programovacím jazyce Java. Nástroje **Android SDK** (Software Development Kit) kompilují kód aplikace, spolu s případnými daty a jinými zdroji, do archivu s příponou *.apk*. Veškerý kód v takovémto archivu je považován za jednu aplikaci. Ta je na zařízení z tohoto balíčku nainstalována.

Jakmile je aplikace nainstalována, každá aplikace pak žije ve svém bezpečnostním sandboxu:

- Operační systém Android je víceuživatelský Linuxový systém, ve kterém je každá aplikace jiným uživatelem.
- Každá aplikace má své unikátní identifikační číslo. Systém nastaví práva všem souborům v aplikaci tak, aby byly přístupné pouze pro danou identitu.
- Každý proces má svůj vlastní virtuální stroj, takže aplikace běží izolovaně od ostatních aplikací.
- Každá aplikace běží ve svém vlastním Linuxovém procesu. Android spustí proces, pokud je potřeba provést některou z částí aplikace. Ukončí jej, pokud již není nadále potřeba nebo v případě potřeby obnovit paměť pro jiné aplikace.

Tímto způsobem Android implementuje princip nejmenších práv. V základním nastavení má každá aplikace přístup pouze ke komponentám potřebným pro svůj chod. Aby získala přístup k dalším částem systému, musí požádat o povolení. Tento systém tak vytváří velmi bezpečné prostředí.

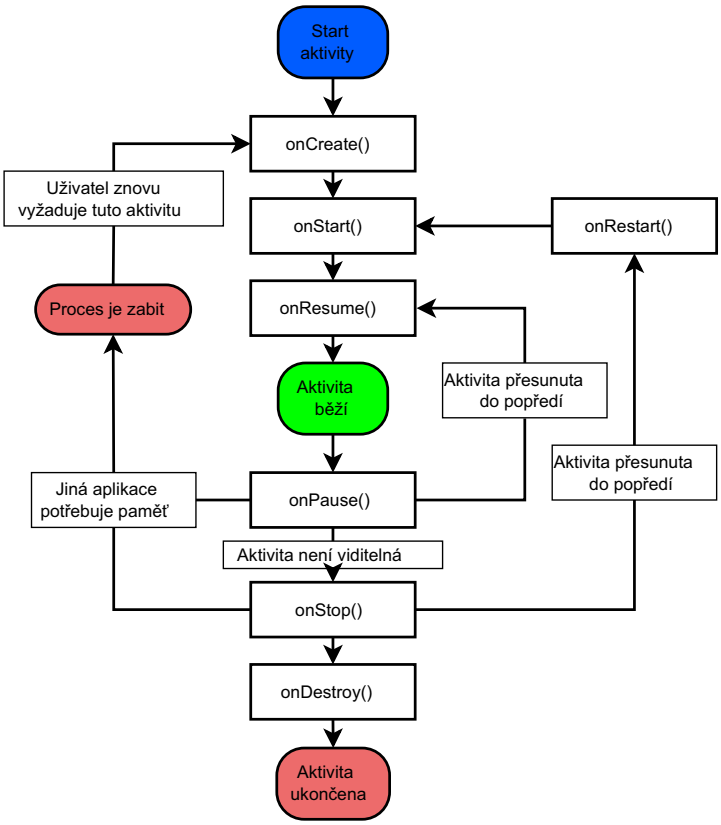
Komponenty aplikací

Aplikace na platformě Android mají čtyři základní stavební komponenty - aktivity, služby, content providers (poskytovatelé obsahu), broadcast recievers. Každá z nich představuje jiný způsob, kterým může systém přistupovat k aplikaci. Ne všechny slouží pouze jako vstupy pro uživatele, ale každá má své opodstatnění a slouží jiným účelům. V následujícím textu budou rozebrány podrobněji.

- **Aktivita**

  Aktivita reprezentuje obecně jednu obrazovku aplikace s uživatelským rozhraním. Například ve hře může být aktivitou počáteční animace, obrazovka s menu, obrazovka

s nápovědou atd. Přesto, že dohromady tvoří jednotlivé aktivity celistvou, uživatelsky přívětivou aplikaci, jsou na sobě nezávislé. Díky tomu je možné spustit z jedné aplikace libovolnou aktivitu jiné aplikace, pokud to povoluje. Například aplikace pro posílání SMS může při tvorbě MMS spustit aktivitu fotoaparátu a získat tak fotografii do přílohy. Následující diagram 2.2 zobrazuje životní cyklus aktivity v systému.



Obrázek 2.2: Životní cyklus aktivity

Životní cyklus aktivity má tři základní stavy vzhledem k systému. V prvním z nich je aktivita spuštěná, zobrazená na obrazovce a provádí kód, případně čeká na uživatelský vstup. Je to stav mezi voláním *onResume()* a *onPaused()*. Další stav je takový, ve kterém je aktivita pozastavena. V tomto stavu si stále uchovává potřebná data, uživatel k ní má přístup, ale není na ni nastaven "focus". V této fázi je většinou překryta jinou aktivitou a v případě nedostatku paměti pro aktivní procesy může být ukončena. Jedná se o fázi mezi voláním *onPaused()* a *onStop()*. V poslední fázi je aplikace zastavena a uživatel k ní ztrácí přístup, dokud není opět spuštěna. Stále si ale uchovává potřebná data, dokud není systémem ukončena. To se může stát brzy, protože má aktivita v této fázi nízkou prioritu a pro ukončení má přednost před aktivitami pozastavenými. I při ukončení aktivity existují mechanismy, jak uchovat potřebná uživatelská data.

- **Služba**

  Služba je komponentou běžící na pozadí, určena k provádění dlouhotrvajících oparací

nebo k práci pro vzdálené procesy. Služba, na rozdíl od aktivity, nenabízí uživatelské rozhraní. Typickou ukázkou služby je hudební přehrávač, který přehrává hudbu na pozadí, zatímco uživatel pracuje s jinou aplikací. Aktivita může službu spustit nebo se k ní navázat a komunikovat s ní.

Na rozdíl od aktivity je životní cyklus služby jednodušší. Služba může buď běžet nebo být ukončena. Ukončit se může sama nebo je možné ji ukončit manuálně, vzdáleně.

- **Content provider**

  Content provider se stará o sdílená data aplikací. Ta mohou být uložena v souborovém systému, v SQLite databázi, na internetu nebo na jiném, aplikacím přístupném, zdroji. Skrze content provider mohou ostatní aplikace k datům přistupovat nebo je i měnit. Například systém Android poskytuje content provider umožňující přístup k uživatelským kontaktním údajům. Díky tomu k nim může (s potřebným povolením) přistupovat libovolná aplikace.

- **Broadcast receiver**

  Broadcast receiver je komponenta, která reaguje na celosystémové broadcastové zprávy. Systém může broadcastem posílat zprávy o zhasnutí obrazovky, nedostatečně nabité baterii, o pořízení snímku fotoaparátem atd. Broadcastové zprávy mohou být generovány i na straně aplikace.

  Přestože broadcast receiver neumožňuje implementaci uživatelského rozhraní, může vytvářet upozornění na stavovém řádku. Obecně je implementován tak, aby prováděl co nejméně práce, například jen vyvolání upozornění a spuštěné nějaké služby.

Aktivace tří ze čtyř komponent (aktivita, služba a broadcast receiver) probíhá s využitím mechanizmu zvaného "*intent*". Jedná se o asynchronní zprávu, která umožňuje propojení dvou komponent za běhu.

- **Manifest aplikace**

  V tomto případě se nejedná přímo o aplikační komponentu, přesto však o nedílnou součást každé aplikace pro operační systém Android, která s komponentami aplikací úzce souvisí. Před spuštěním každé aplikační komponenty musí operační systém zjistit, zda tato komponenta existuje. Tuto informaci nalezne v manifestu aplikace, XML souboru s názvem **AndroidManifest.xml**, kde musí být uvedeny všechny komponenty aplikace. Manifest je nezbytnou součástí každé aplikace a kromě deklarování komponent aplikace obsahuje následující:

  - Název balíku aplikace, který slouží jako její jednoznačný identifikátor.
  - Vstupní komponentu aplikace (např. MDSplashActivity v případě mé aplikace).
  - Seznam aplikací požadovaných uživatelských oprávnění, jako třeba přístup k Internetu nebo ke kontaktům.
  - Deklaruje minimální verzi API požadovanou aplikací.
  - Deklaruje hardwarové a softwarové požadavky aplikace (přístup k fotoaparátu, ke službě bluetooth, atd.).
  - Seznam potřebných knihoven, které nejsou součástí základního API, jako například knihovna Google Maps.

Listing 2.1 je ukázkou souboru AndroidManifest.xml k aplikaci, která požaduje přístup k fotoaparátu, vstupní aktivitou je MDSplashActivity a obsahuje jednu další aktivitu s názvem MDDetectodActivity:

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.motiondetector">
    <uses-permission android:name="android.permission.CAMERA" />
    <application android:icon="@drawable/icon" android:label="@string/
        app_name">
        <activity android:name=".MDSplashActivity" android:label="
            @string/app_name">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER
                    " />
            </intent-filter>
        </activity>
        <activity android:name="MDDetectorActivity"></activity>
    </application>
</manifest>
```

Listing 2.1: Ukázka souboru AndroidManifest.xml

NDK

Android NDK je sada nástrojů, která umožňuje použití komponent využívajících nativního kódu. Android aplikace běží ve virtuálním stroji Dalvik a jsou psány v jazyce Java. NDK dovoluje implementovat části aplikací v nativním kódu v jazyce C nebo C++ a umožňuje tak znovupoužití již napsaného v těchto jazycích a hlavně v některých opodstatněných případech přináší urychlení zpracování. Vývojářům nabízí:

- Sadu nástrojů na kompilaci nativních knihoven napsaných v C/C++.
- Způsob, jak přidat nativní knihovny do distribučního .apk balíčku.
- Sadu nativních systémových hlaviček se zaručenou podporou v budoucích verzích.

Nejnovější verze NDK (údaj z března 2011) nabízí podporu pro instrukční sadu ARM procesorů ve verzích ARMv5TE a ARMv7-A. Do budoucna je plánována podpora pro instrukční sadu procesorů x86.

Strojový kód ARMv5TE bude běžet na všech Android zařízeních s ARM procesory. ARMv7 bude použitelný pouze na zařízeních s kompatibilním CPU. Hlavním rozdílem je, že ARMv7 podporuje hardwarovou FPU (Floating Point Unit), Thumb-2 a NEON instrukce. Při překladu je možno vybrat jednu z verzí nebo překládat pro obě.

- **Vhodné použití nativního kódu**

  V současné době není využití nativního kódu pro většinu aplikací žádným přínosem. Je to proto, že v mnoha aplikacích by jeho využití pouze zvýšilo nároky na implementaci, ale výsledná aplikace by nedosahovala většího výkonu.

  Vhodnými kandidáty na využití schopností NDK jsou výpočetně náročné operace s vysokými požadavky na paměť jako zpracování signálů, simulace fyzikálních jevů

atd. Další výhodou je možnost efektivního znovuužití velkých částí existujícího kódu napsaného v jazyce C/C++. V této práci jsem tak využil možnosti relativně snadno přeložit knihovnu OpenCV a těžit z jejich rozsáhlých schopností.

Platforma Android nabízí dva způsoby k použití nativního kódu:

1. Napsání aplikace s využitím rozhraní Android a použití **JNI** (*Java Native Interface*) k přístupu k API poskytovaného Android NDK. Výhodou je možnost využívat pohodlného vývoje pro Android se zachováním možnosti přistupovat k nativním metodám. Tohoto způsobu jsem využil i já při implementaci aplikace k této práci (jedním z hlavních důvodů byla také nemožnost přistoupit ke kameře přímo z nativního kódu).
2. Napsání nativní aktivity, která umožňuje implementaci všech metod životního cyklu aktivity. To je možné pouze pro zařízení se systémem Android 2.3 a vyšším.

- **Obsah NDK**

  Distribuce NDK obsahuje API, dokumentaci ve formátu HTML a několik vzorových aplikací pro lepší pochopení práce s nativním kódem.

  NDK zahrnuje sadu nástrojů pro kompilaci, linkování, atd. které mohou generovat výsledné binární soubory pro ARM procesory. Tyto nástroje jsou dostupné pro operační systémy Linux, OS X a Windows (s prostředím Cygwin). Poskytuje také hlavičkové soubory k stabilnímu nativnímu API, u kterého je garantována podpora pro další verze. Patří sem:

  - libc (C knihovna)
  - libm (početní knihovna)
  - hlavičky pro rozhraní JNI
  - libZ (knihovna pro kompresi)
  - liblog (logování pro Android)
  - OpenGL ES 1.1 a 2.0 (3D grafické knihovny)
  - libjnigraphics (přístup k pixel bufferu)
  - minimalistická sada hlaviček k podpoře C++
  - OpenSL (nativní audio knihovny)
  - API pro nativní aplikace

2.1.3 Hardware

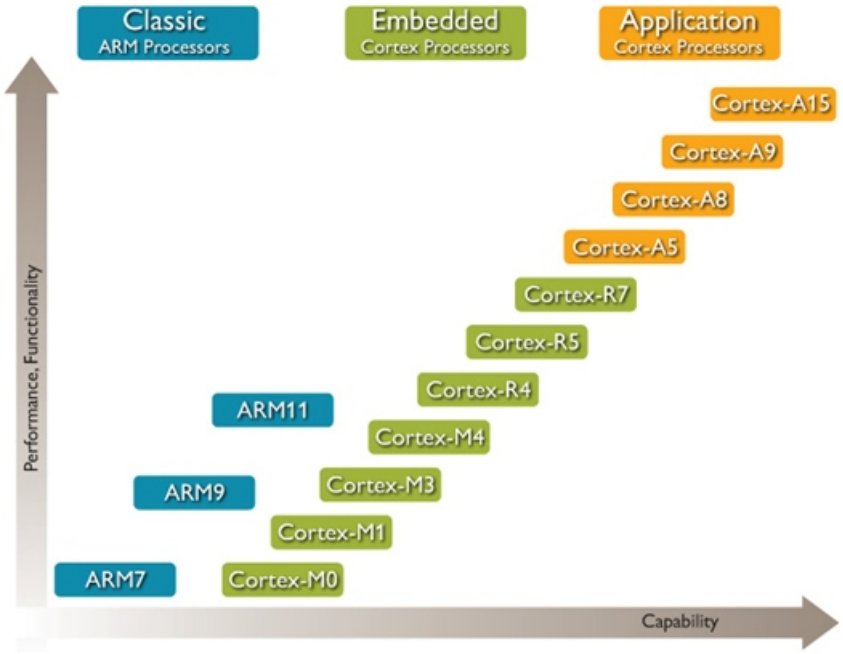
ARM

ARM je 32bitová RISK architektura procesorů vyvinutá společností ARM Holdings. Zkratka ARM pochází z dřívějšího označení Advanced RISC Machine nebo také Acorn RISC Machine. V počtu prodaných kusů je Architektura ARM nejvíce používanou architekturou s 32bitovou šířkou slova (v roce 2009 mělo přibližně 90% prodaných 32bitových RISC vestavěných systémů alespoň jeden ARM procesor).

Architektura ARM je licencovaná. Společnost ARM Holdings je vývojářem architektury a prodává výrobní licence mnoha jiným společnostem (např. Freescale, LG, Samsung, Texas Instruments, Qualcomm a mnohé další).

Důvodem, proč se tato architektura uchytila nejvíce v oblasti vestavěných systémů je její nízká spotřeba energie při zachování vysokého výpočetního výkonu, což jsou důležité parametry pro zařízení napájená bateriemi. Přesto se v dnešní době objevují procesory ARM i v jednodušších osobních počítačích. [2]

Na obrázku 2.3 je základní přehled jader procesorů ARM v závislosti na možnostech a výkonu, rozdělený do barevných kategorií podle užití.



Obrázek 2.3: Přehled jader architektury ARM. Převzato z [3]

V této podkapitole dále popíši základní vlastnosti procesorů z rodiny Cortex-A s architekturou ARMv7-A, s jádrem Cortex-A8.

Cortex-A8

Série Cortex-A patří do řady aplikačních procesorů. Nabízí širokou škálu nástrojů pro cenově dostupná zařízení hostující plnohodnotný operační systém a uživatelské aplikace. Tyto procesory jsou vhodné pro užití v chytrých mobilních telefonech (smartphonech), přenosných výpočetních platformách, set-top boxech, tiskárnách, atd.

Cortex-A8 je aplikační procesor založený na architektuře ARMv7. Mezi jeho hlavní rysy patří:

- Frekvence procesoru od 600MHz až přes 1GHz.
- Vysoký výkon, superskalární architektura.
- Technologie NEON (viz. 2.1.3) pro zpracování multimédií.
- A další (popsáno v [16]).

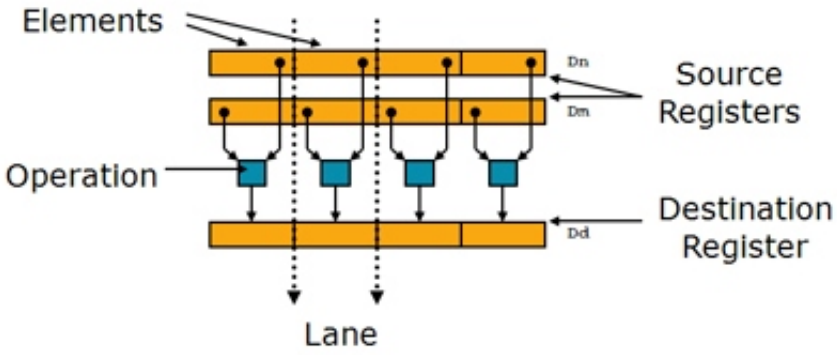
Procesory typu Cortex-A8 nalezneme v mnoha mobilních zařízeních s různými operačními systémy. Z těch nejoblíbenějších, prodávaných s OS Android, jsou to chytré mobilní telefony (smartphony) HTC Desire, HTC Evo 4G, Motorola Droid, Samsung Galaxy S, Sony Ericsson Xperia X10 a další, z tabletů potom například oblíbený Samsung Galaxy Tab.

NEON

Tento koprocesor je založen na architektuře **SIMD** (Single Instruction, Multiple Data). Jeho cílem je urychlení multimediálních výpočtů pomocí vektorového zřetězeného zpracování aritmetických instrukcí. Je oddělen od jádra procesoru a má vlastní přístup do paměti cache (tj. vlastní load/store jednotku).

Na obrázku 2.4 je nastíněna činnost jednotky při zpracování instrukcí. Registry považujme za vektory nebo prvky stejného datového typu (může se jednat o signed/unsigned 8-bit, 16-bit, 32-bit, 64-bit nebo single precision floating point[1]) Instrukce provádí stejnou operaci ve všech pruzích (lanes).

Koprocesor NEON obsahuje jednu jednotku pro celočíselné výpočty a jednu jednotku pro výpočty v plovoucí řádové čárce. V praxi je zároveň s aritmetickou instrukcí často vydávána druhá instrukce pouze pro vstupní/výstupní operaci, která je nezávislá na výpočetních jednotkách. Vlastní výpočetní jednotky mají šest zřetězených stupňů, takže v jednom taktu lze mít rozpracováno až dvanáct instrukcí. [16, 3]



Obrázek 2.4: Vektorové zpracování na jednotce NEON. Převzato z [3]

2.2 OpenCV

OpenCV je multiplatformní open source knihovna pro počítačové vidění napsaná v jazyce C a C++. Knihovna byla navržena tak, aby se z ní stal efektivní a rychlý nástroj použitelný při vývoji real-time aplikací. Největší využití nalézá v oblasti počítačového vidění, v bezpečnostních a monitorovacích systémech a v umělé inteligenci, zejména v interakci stroj - člověk.

Knihovna nabízí velké množství funkcí pro práci s obrázky nebo jejich sekvencemi (videem). Jde například o vytváření a ukládání obrázků, kreslení základních geometrických

[1]Standard IEEE 754 jej definuje jako 32 bitové číslo které má 1 bit znaménkový, 8 bitů určených pro exponent a 23 bitů pro mantisu.

útvarů a textu, aplikaci široké škály filtrů, převody mezi barevnými modely, matematické operace nad jedním a více obrázky. Umožňuje také snadné připojení kamery.

Množina nabízených funkcí mi usnadnila implementaci cílového programu a mohl jsem se tak více zaměřit na návrh samotného algoritmu detekce a lokalizace pohybu a jeho optimalizaci pro různá prostředí. Informace čerpány z [11, 6].

2.3 Použité algoritmy detekce pohybu

Detekovat pohybující se objekty v prostoru je pro lidské oko jednoduchým úkolem, který bere člověk jako samozřejmost. Díky vlastnostem lidského vidění a díky zkušenostem získaným během života, je člověk schopen nejen rozeznat pohyb, ale dokonce i odhadnout vzdálenost a rychlost pohybujícího se předmětu. Z vývojového hlediska jde o dovednosti nezbytné pro přežití.

Díky moderním technologiím a rostoucímu výpočetnímu výkonu počítačů jsme v dnešní době schopní různými způsoby detekovat pohyb i pomocí strojů. Tato práce je zaměřena na detekci pohybu ve video sekvenci. Tento problém není tak triviální, jak by se mohl zdát.

Detekce pohybujících se objektů je zajímavým problémem, který nachází uplatnění například v bezpečnostních systémech, v oblastech, kde je potřeba sledovat určitý provoz (například doprava, stanice hromadných dopravních prostředků, ...), ovládání zařízení pomocí gest nebo jako vstup pro další práci s obrazem (např. rozpoznávání osob).

Ve své práci jsem se zaměřil na detekci jednotlivých částí v obraze, jejichž změny v čase signalizují možný pohyb. Po lokalizaci těchto částí je zvýrazním a v rámci vyvíjené aplikace s nimi budu nakládat dle potřeby.

Základním algoritmem používaným s dobrými výsledky pro detekci pohybu ve video sekvenci, je metoda **odečítání pozadí** [6], která porovnává změny v po sobě jdoucích obrázcích. Pro dobré výsledky této metody je ale potřeba obrázky vhodně upravit, aby se zamezilo nežádoucím vlivům plynoucím z parametrů a nedostatků snímače videa a zrychlil se výpočet těchto obecně výpočetně náročných operací. Konkrétně se jedná o vhodnou volbu barevných modelů, o filtraci vstupního obrazu a o redukci barevného prostoru. Všemi těmito problémy se po teoretické stránce zabývá tato podkapitola.

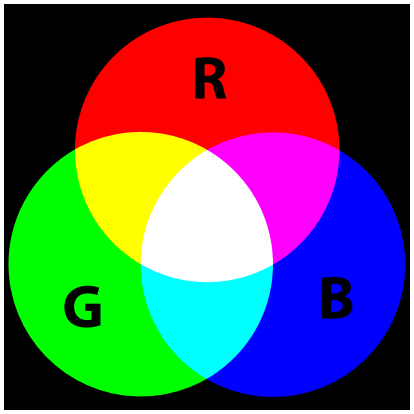
2.3.1 Barevné modely

Barevné modely nám umožňují pracovat s barvami a míchat je. Existuje několik druhů barevných modelů, které jsou vytvořeny tak, aby co nejlépe vyhovovaly jejich účelu užití. Jsou přizpůsobeny uživatelům (grafik, programátor), technologiím a oblastem použití (tisk, televizní technika, videotechnika). Pro účely této práce je stěžejní pochopení následujících tří barevných modelů: RGB, YUV a grayscale neboli šedé tóny (o grayscale více v kapitole 2.3.2) [13].

RGB

RGB model je zřejmě nejpoužívanějším barevným modelem u obrázkových formátů. Jedná se o aditivní skládání barev, kde každá barva vznikne přidáním různého množství červené, zelené a modré barvy k barvě černé. Je-li množství všech tří složek nulové, získáme barvu černou, naopak pokud je maximální, dostaneme barvu bílou. Každá barva bývá nejčastěji reprezentována trojicí hodnot (hodnota červené složky, hodnota zelené složky, hodnota modré složky), např.: (255, 255, 255) pro bílou barvu. Jsou-li všechny tři hodnoty stejné,

např.: (127, 127, 127), získáme odstín šedi. Model aditivního míchání barev je znázorněn na obrázku 2.5.



Obrázek 2.5: Aditivní míchání barev u RGB modelu. Převzato z [12]

Samotný RGB model může mít různé digitální reprezentace [12]. Ty se od sebe liší binární reprezentací jednotlivých složek. Obvykle je každá složka reprezentována celým číslem v rozsahu od 0 do $2n - 1$. Obecně nejčastější reprezentací je 24 bitová s 8 bity pro každou barevnou složku, jinak zvaná také *Truecolor* (obr. 2.6).

Sample Length:	8								8								8							
Channel Membership:	Red								Green								Blue							
Bit Number:	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Obrázek 2.6: 24 bitová digitální reprezentace RGB modelu. Převzato z [12]

V implementaci detektoru pohybu pro platformu Android jsem využil jako výstup RGB model s označením 565, jinak nazývaný také *Highcolor* (obr. 2.7). Jedná se o 16 bitovou reprezentaci, kde červená a modrá složka mají velikost 5 bitů a zelená využívá o jeden bit více (je to z toho důvodu, že lidské oko vnímá jas zelené komponenty v největší míře).

Sample Length:	5					6						5				
Channel Membership:	Red					Green						Blue				
Bit Number:	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
RGBAX	R. G. B. A. X															
Sample Length Notation:	5.6.5.0.0															

Obrázek 2.7: 16 bitová digitální reprezentace RGB modelu 565. Převzato z [12]

YUV

Model YUV je poněkud odlišný od ostatních kolorimetrických modelů. Jedná se v podstatě o lineární transformaci obrazových dat RGB a je široce využíván k zakódování barvy u televizního přenosu. Y-signál určuje úroveň šedi nebo jas, U-signál a V-signál nesou informaci o barvě. Výstupní obraz z kamery na platformě Android je také ve formátu YUV a proto je nutné jej převést na nějaký formát vhodnější k práci.

Dle [4] jsou vzorce 2.1, 2.2 a 2.3 obecnými vzorci pro převod z formátu YUV na jednotlivé RGB komponenty.

$$B = 1.164(Y - 16) + 2.018(U - 128) \tag{2.1}$$
$$G = 1.164(Y - 16) - 0.813(V - 128) - 0.391(U - 128) \tag{2.2}$$
$$R = 1.164(Y - 16) + 1.596(V - 128) \tag{2.3}$$

2.3.2 Předzpracování obrazu

Jak již bylo řečeno výše, předzpracování obrazu je nezbytnou částí detektoru, která má za úkol odstranit nebo potlačit nedostatky video sekvence způsobené zejména nedokonalostí snímacího čipu a snížit výpočetní nároky aplikace. Následuje teoretický popis metod, které jsem použil při implementaci problému.

Redukce barevného prostoru na stupně šedi (grayscale)

Převod na stupně šedi se řadí do kategorie redukce barevného prostoru, kdy je barevný obrázek převeden a následně reprezentován odstíny šedé barvy (viz obr. 2.8). Nevýhodou tohoto modelu je ztráta informace o barvě, nicméně v různých oblastech počítačového vidění si vystačíme pouze s intenzitou, kterou tento model zachovává. Hlavní výhodou je potom podstatně nižší velikost výsledného obrázku (obecně 1/3 oproti RGB obrázku), neboť k reprezentaci jednoho pixelu nám stačí většinou pouze 8 bitů (256 úrovní šedi).

Pro převod z RGB modelu se používá vzorce 2.4.

$$I = 0,299R + 0,587G + 0,114B \tag{2.4}$$

R značí barvu červenou, G barvu zelenou a B modrou barvu původního transformovaného snímku. I je výsledná intenzita ve stupni šedi. Tyto poměry jsou stanoveny z odlišné citovosti lidského zraku na základní barvy. Lidské oko je nejvíce citlivé na zelenou barvu, proto je její podíl největší. [14]

Prahování

Prahování (ang. thresholding) je metodou kategorizace obrazových pixelů, která umožňuje určit hranici intenzity v obrazu ve stupních šedi a odstranit, případně zachovat, pixely, které pod danou hranici nespadají. Základní myšlenkou je možnost zpracování pouze takových pixelů, které splňují kritéria intenzity. [6]

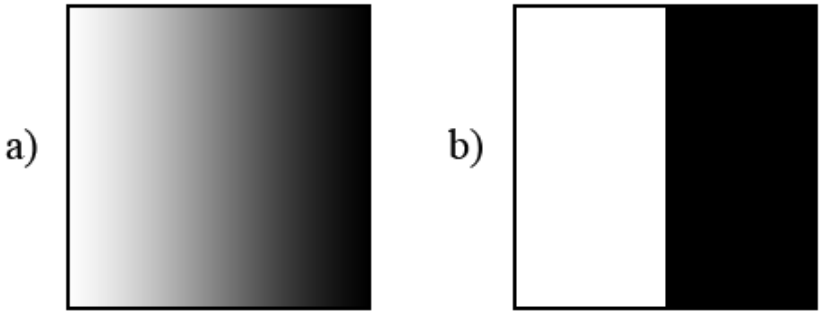
Matematicky lze prahování vyjádřit následovně: Máme-li vstupní obraz $O(x, y)$ a práh P, tak pro výstupní binární obraz $B(x, y)$ platí rovnice 2.5.

$$B(x,y) = \begin{cases} 1 & \text{pro } O(x,y) \geq P \\ 0 & \text{pro } O(x,y) < P \end{cases} \tag{2.5}$$

Obrázek 2.8: Redukce barevného prostoru. Převzato z [14]

Prakticky tak díky prahování můžeme určovat citlivost detektoru. Zvolením vhodného prahu určíme, jak podstatné změny v obraze považujeme za zajímavé a případné menší změny pak již neuvažujeme. Problémem ale zůstává zvolení vhodného prahu, který se může pro různé podmínky lišit.

Příklad tohoto filtru demonstruje obrázek 2.9: vstupem filtru je obrázek a), prahováním s hodnotou 128 vznikne výstupní obrázek b).



Obrázek 2.9: Prahování s hranou o hodnotě 128

Mediánový filtr

Mediánový filtr patří do kategorie *vyhlazování* (ang. smoothing) nebo také *rozmazávání* (ang. blurring). Je to jednoduchá a často používaná technika v oblasti zpracování obrazu. K použití vyhlazování může být mnoho důvodů, ale obvykle se používá k redukci šumu. Využití nachází také při změně rozlišení obrazu.

Mediánový filtr pracuje tak, že každý pixel nahradí střední hodnotou pixelů v jeho okolí (obvykle se používá okolí 3x3 pixely). Tento filtr lze aplikovat na monochromatické obrázky, obrázky ve stupních šedi i na obrázky barevné, nemůže však být aplikován na místě[2]. Výsledkem je, že pokud se na obrázku vyskytují osamocené rozdílné pixely, přizpůsobí se

[2]Při filtraci je potřeba pomocného obrázku pro ukládání mezivýsledků jednotlivých pixelů.

okolí a splynou (viz. obr. 2.10). Vedlejším efektem je mírné rozmazání obrázku. [7] [6]

Obrázek 2.10: Mediánový filtr. Převzato z [7]

Eroze

Erozní filtr pracuje na podobném principu, jako mediánový filtr. Na rozdíl od něj se ale používá většinou v monochromatických obrázcích. Výsledná barva každého pixelu je zde také počítána v závislosti na okolí, nicméně místo mediánu je výslednou hodnotou nejmenší hodnota v okolí. Výsledkem eroze bývá poněkud tmavší obrázek. Bílý šum je odstraněn a bílé plochy jsou poněkud zmenšeny (viz. obr. 2.11). Filtr nelze aplikovat na místě. [7]

Obrázek 2.11: Erozní filtr. Převzato z [7]

Dilatace

Dilatace pracuje na stejném principu jako eroze s tím rozdílem, že v okolí vybírá pixely s největší hodnotou. Výsledkem dilatace je tedy poněkud světlejší obrázek. Bílé plochy jsou zvětšeny (viz. obr. 2.12). Užívá se po předchozích filtrech, aby vrátila světlé plochy do původní velikosti. [7]

Obrázek 2.12: Dilatační filtr. Převzato z [7]

2.3.3 Odečítání pozadí

Metoda *odečítání pozadí* (ang. background subtraction) patří k nejjednodušším metodám detekce pohybu. Přes svou jednoduchost však dosahuje velmi dobrých výsledků co se úspěšnosti i efektivity týče. Informace k následujícímu popisu metody jsou čerpány z [6] [9] [7].

Principy metody

Hlavním cílem této metody je snaha odlišit od sebe **pozadí** a **popředí** snímané scény v jednotlivých snímcích video sekvence.

Pozadí je těžko definovatelné, protože se liší v závislosti na účelu aplikace. Obecně se jedná o statickou scénu, která se v průběhu video sekvence nemění. Buď neobsahuje žádné pohyblivé objekty (snímáme-li například provoz na dálnici, je vhodné, aby jako pozadí byla vnímána prázdná silnice) nebo může obsahovat objekty periodicky se měnící (např. kyvadlové hodiny v pokoji, který je monitorován).

Popředí jsou objekty, které jsou předmětem našeho zájmu (pohybující se objekty). Popředí získáme, jak už název metody napovídá, odečtením pozadí od aktuálního snímku.

Abychom mohli tuto metody používat, je potřeba mít k dispozici model pozadí tzv. referenční snímek. Prvním krokem je právě jeho vytvoření. Nejjednodušší metodou je vybrat snímek video sekvence, u kterého předpokládáme, že je snímaná scéna v klidovém stavu, tzn. ve stavu, kdy se v ní nic nehýbe. Odlišnosti od tohoto referenčního snímku budou dále označovány detektorem jako pohyb. Na obrázku 2.13 můžeme vidět jednotlivé výše zmiňované části. Vlevo pozadí, uprostřed aktuální snímek a vpravo popředí za ideálních podmínek.

Slabiny odečítání pozadí

Přestože zmíněná metoda odečítání pozadí funguje dobře pro jednoduché scény, má i své nedostatky. Většina z nich pramení z toho, že jsou všechny pixely v obraze uvažovány jednotlivě. Kvůli tomu, že při porovnávání nezohledňuje okolní, je například podmínkou statické pozadí. Každá změna zorného úhlu kamery nebo i změna v osvětlení scény může zamezit správnému chování detektoru.

Obrázek 2.13: Odečítání pozadí. (Zleva: pozadí, aktuální snímek, popředí)

V základní neupravené podobě by tak tato metoda byla použitelná pouze v místnosti se stálým osvětlením a se snímačem v dokonale stabilizované poloze. Tyto problémy jsou do jisté míry redukovatelné průběžnými aktualizacemi pozadí, nicméně způsob aktualizace pozadí je závislý na prostředí scény (měnící se osvětlení v průběhu dne, pohyb stromů ve větru, pohyb mraků po obloze, atd.) a ovlivňuje výslednou kvalitu detekce.

Můj detektor předpokládá nehybný snímač kamery a aktualizace pozadí je řešena tak, že každý nový snímek je v jisté míře připočítá k současnému pozadí. Tato míra je nastavitelná, ale řádově se pohybuje v setinách až desetinách procenta. Vedlejším jevem však je, že pokud se objekt zastaví ve scéně, po určité době splyne s pozadím. To však může v jistých ohledech znamenat i přínos (například auto přijedete na parkoviště a po nějaké době se stane součástí pozadí).

Algoritmy odečítání pozadí

Pro porovnání zmíním dva možné algoritmy odečítání pozadí od jednotlivých snímků. Prvním z nich je jednodušší porovnání jasového diagramu a druhým je porovnání pixelů. Každý z těchto postupů má své uplatnění a také výhody i nevýhody.

- **Porovnání jasového diagramu**

  Jedná se o velmi jednoduchou metodu spočívající v porovnání jasové složky snímků. Nejprve vypočteme histogram pozadí. Histogram může být implementován např. jako pole, jehož počet prvků se rovná počtu možných jasových intenzit v obrázku. Prvky pole jsou počty pixelů ve snímku majících stejnou intenzitu, jako je index prvku. Jasovou hodnotu každého pixelu z RGB obrázku získáme převodem na odstíny šedi.

  Porovnání aktuálního snímku s referenčním provedeme odečtením histogramů. Výsledkem bude pole obsahující počty odlišných pixelů. Pohyb může být detekován například při dosažení zadané maximální hodnoty součtu polí histogramu.

  Tento algoritmus je velmi rychlý a ponechává nám dostatek výpočetního výkonu pro aplikaci množství filtrů pro předzpracování obrazu. Jeho hlavní nevýhodou ale je, že neumožňuje lokalizovat pohyb ve snímku.

- **Porovnání pixelů**

  Na rozdíl od předchozího přístupu nám algoritmus porovnávání jednotlivých pixelů umožňuje snadnou lokalizaci pohybu ve snímku. To s sebou ovšem nese větší výpočetní a paměťovou náročnost, což je třeba vzít v úvahu v souvislosti s možnými

úspornými opatřeními. Pro aplikaci jsem zvolil tento algoritmus, protože se správně zvolenou metodou aktualizace pozadí dosahoval při průběžném testování velmi dobrých výsledků.

Nejvýraznějším úsporným opatřením, které v aplikaci používám, je redukce barevného prostoru snímků. Díky převodu z tříbarevných obrázků na obrázky ve stupních šedi snížíme velikost zpracovávaných snímků na jednu třetinu.

Při vhodném použití filtrů se tento algoritmus dokáže dobře vyrovnat se šumem a anomáliemi vzniklými chybami snímače. Vyžaduje však použití pomocných masek pro filtrování. Hlavní nevýhodou je jeho výše zmíněná vyšší náročnost.

Kapitola 3

Návrh a implementace

Tato kapitola se věnuje samostatné implementaci aplikace pro demonstrační účely. Pro rychlejší vývoj a snadnější optimalizaci jádra detektoru pohybu byla nejdříve vytvořena aplikace pro osobní počítač. Aplikace byla napsána v jazyce C a využívala knihovnu OpenCV. Po ověření funkčnosti detektoru bylo její jádro portováno do nativního kódu a s drobnými úpravami využito jako knihovna ve výsledné mobilní aplikaci. V následujících podkapitolách se věnuji struktuře aplikace a implementaci jednotlivých dílčích celků.

Zdrojem informací o vývoji aplikací pro platformu Android mi byla kniha [8] a oficiální podpora pro vývojáře [1].

3.1 Jádro aplikace

Kromě samotného jádra detektoru a funkcí napsaných v jazyce C, kterým se budu věnovat níže a tříd, které implementují uživatelské rozhraní na úrovni aktivit, obsahuje aplikace několik důležitých tříd, které si zaslouží zvláštní pozornost.

- **Třída MDActivity**

  Třída *MDActivity* je rodičovskou třídou všech aktivit aplikace. Její hlavní úlohou je uchovávat a zprostředkovávat parametry a nastavení napříč celou aplikací. Uchovává jak nastavení chování aplikace, tak nastavení parametrů detektoru. Ty lze měnit v aktivitě pro nastavení *MDSettingsActivity*.

- **Rozhraní IEventHandler**

  Toto jednoduché rozhraní slouží posílání zpráv třídám, které jej implementují. V aplikaci jej implementuje třída *MDDetectorActivity*. Je tak možné ji asynchronně informovat o tom, že se vyskytl pohyb v obraze.

- **Třída OpenCV**

  Tato třída je hlavní prostředníkem mezi jádrem detektoru v nativním kódu a zbytkem aplikace. Načítá nativní knihovnu *libopencv.so* a definuje hlavičky jednotlivých nativních funkcí. Třída si nese seznam posluchačů (*Set<IEventHandler> listeners*) implementujících rozhraní *IEventHandler*, které informuje v případě, že z nativního kódu, obdrží informaci o detekovaném pohybu (zavoláním funkce *motionDetected(double percentage)* z nativního prostředí).

- **Třída MDDetectorActivity**

  Třída *MDDetectorActivity* implementuje hlavní aktivitu aplikace zobrazující detekovaný prostor včetně výstupu detektoru a umožňující nastavení **ROI** (Region Of Interest - oblast zájmu). Tato aktivita se stará o inicializaci detektoru podle nastavení, zobrazení výstupu detektoru, blokování přechodu telefonu do režimu spánku, oznámení pohybu (odeslání SMS, oznámení na obrazovce telefonu, zvukový alarm) a nastavení/rušení ROI.

  Třída využívá *SurfaceView MDPreview* pro zachytávání videa z kamery, to následně předává svému *View MDDetectorView*, které nechává jednotlivé snímky video sekvence zpracovat knihovnou *libopencv.so* a vykresluje je na své plátno.

  Volba ROI je implementována následovně: nad náhled z kamery je vykreslen obdélník s hranami o velikosti 1/3 šířky/délky obrazovky. Třída implementuje metodu *onTouchEvent()*, která naslouchá dotekům obrazovky. V případě položení prstu na obrazovku je vybrán nejbližší roh obdélníku. Ten se následně pohybuje ve směru pohybu prstu po displeji. Roh obdélníku zůstane na pozici, kde se prst opět zvedne z obrazovky. Finální výběr je potřeba potvrdit tlačítkem v nabídce menu.

3.2 Nativní kód

Veškerý nativní kód je implementován v knihovně *libopencv.so*. Ta obsahuje knihovnu OpenCV pro počítačové vidění, funkce na převod snímků mezi barevnými formáty a funkci pro převod *IplImage* (obrazový formát, se kterým pracuje většina funkcí knihovny OpenCV) zpět na bitmapu. Rozhraní mezi Java kódem a nativním kódem je zprostředkováno přes **JNI** (Java Native Interface).

Zdrojový kód knihovny OpenCV spolu s funkcemi na převod mezi bitmapou a IplImage jsem čerpal z [17]. Zdrojový kód pro převod z YUV do RGB je převzat z [5]. Konverzní funkce je implementována jak na straně Java, tak v C a v podkapitole 3.5 bude použita k porovnání výkonnosti náročných operací implementovaných v Javě, nativním kódu bez využití vektorového zpracování (pro procesory bez jednotky NEON) a nativním kódu s využitím NEON.

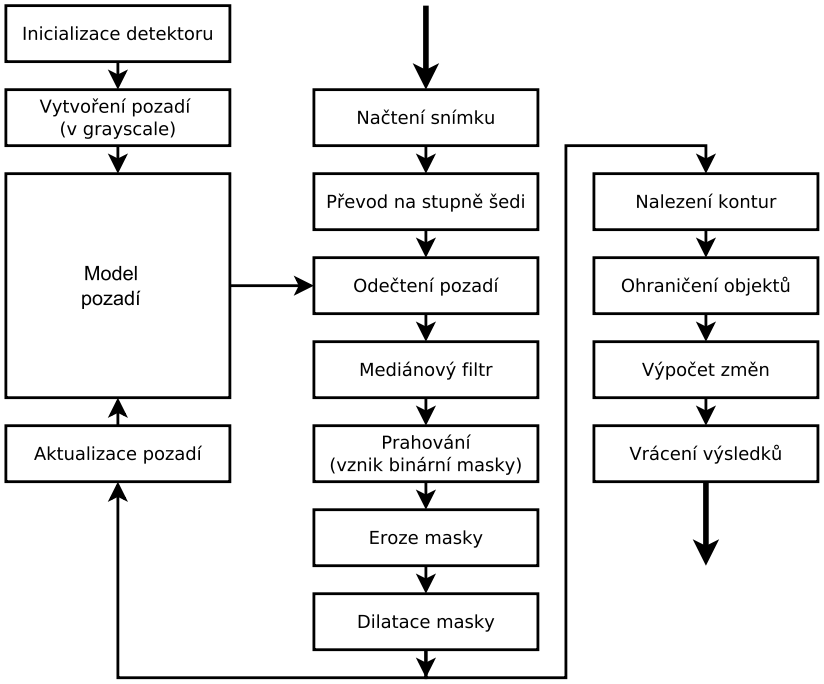
V nativním kódu jsem implementoval celé jádro detektoru. Jeho implementace je popsána níže v podkapitole 3.3.

3.3 Jádro detektoru

Samotné jádro detektoru využívá teorie popsané v podkapitole 2.3. Jádro zpracovávající sekvenci po sobě jdoucích snímků z videa je napsáno v jazyce C a využívá knihovnu pro počítačové vidění OpenCV. Výsledný algoritmus (viz obr. 3.1) vznikl na základě informací získaných v [6] a následným testováním chováním aplikace za různých podmínek.

Jak je z diagramu na obr. 3.1 patrné, detekce začíná inicializací detektoru. Nastaví se parametry a proběhne kalibrace, tj. čeká se, dokud se příchozí snímky neustálí a potom se pořídí referenční snímek pozadí převedený do odstínů šedi pro úsporu prostředků.

Načte se aktuální snímek a po převodu do grayscale se odečte pozadí. Pro odstranění množství šumu vzniklého snímačem je aplikován mediánový filtr a snímek je vyhlazen. Prahováním vznikne binární maska označující případné oblasti pohybu. Hodnota prahu určuje citlivost detektoru. V případě, že je práh příliš nízký, detektor může nesprávně detekovat



Obrázek 3.1: Algoritmus jádra detektoru

i velmi malé změny v osvětlení jako pohyb. V opačném případě zase nastává problém s detekcí pohybujících se objektů, které mají podobné vlastnosti jako pozadí v místě pohybu.

Binární maska většinou stále obsahuje nějaké nečistoty, a proto je na ni aplikován filtr eroze, který odstraní malé objekty, které můžeme považovat za šum. Filtr eroze má jako vedlejší účinek zmenšení detekovaných ploch. Proto je potřeba na masku následně aplikovat dilatační filtr pro nápravu vzniklé škody.

V tomto místě je v kódu provedena aktualizace pozadí využitím funkce *cvRunningAvg()*. Funkce spočítá váženou sumu pozadí a nového snímku tak, že nový snímek je započten jen s určitou vahou. Ta je konfigurovatelná v nastavení a určuje, jak rychle model pozadí zapomíná na předchozí snímky.

Následně jsou aplikovány funkce pro nalezení kontur objektů v binární masce, jejich ohraničení čtyřúhelníky, vykreslení těchto čtyřúhelníků do vstupního snímku a jeho následné odeslání zpět detektoru.

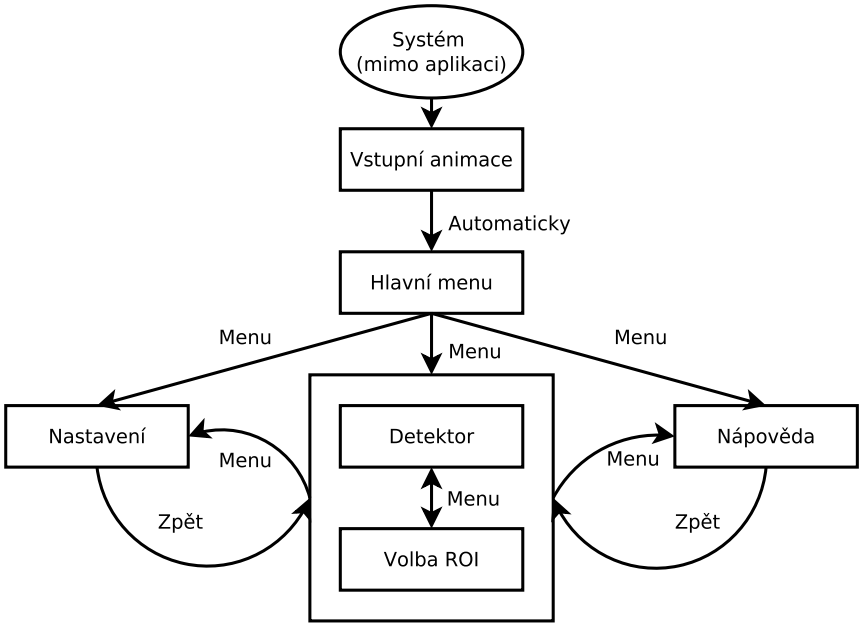
Před odesláním snímku je ještě z upravené binární masky spočteno procento změn ve snímku oproti snímku referenčnímu. V případě nastavení ROI jsou změny počítány pouze v oblasti našeho zájmu. Pokud změny dosahují nastavené hranice, nativní kód zavolá Javovskou funkci s informací o množství změn.

3.4 Uživatelské rozhraní

Obrázek 3.2 znázorňuje jednotlivé části uživatelského rozhraní podle oken, která vidí uživatel. Obdélníky znázorňují jednotlivá okna aplikace. Spojnice značí přechody mezi nimi a způsob, jak je provést. Pro zjednodušení není u každého okna uvedeno tlačítko *Zpět*, které vždy vrací focus na předchozí aktivitu. Systém značí obrazovku mimo aplikaci, tj. místo, ze

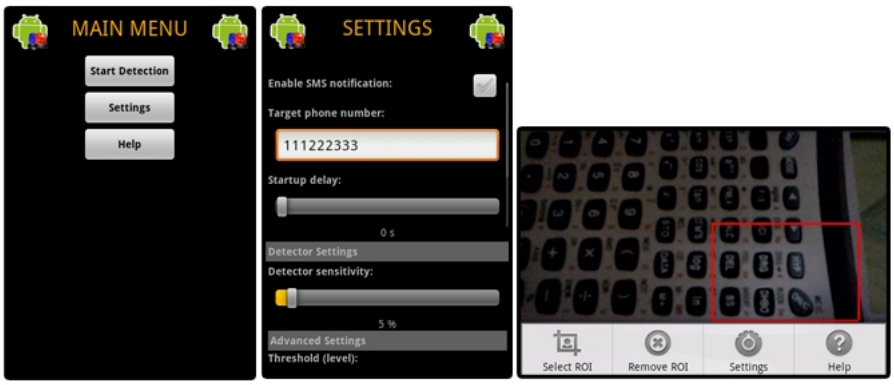
kterého je aplikace spuštěna a kam se vrátí v případě opuštění aplikace stisknutím tlačítka *Domů*.

Jednotlivá okna aplikace jsou implementována jako příslušné aktivity (*MDSplashActivity*, *MDMenuActivity*, *MDSettingsActivity*, *MDHelpActivity*). Okna detektoru a výběru ROI tvoří jednu aktivitu *MDDetectorActivity*.



Obrázek 3.2: Diagram uživatelského rozhraní aplikace

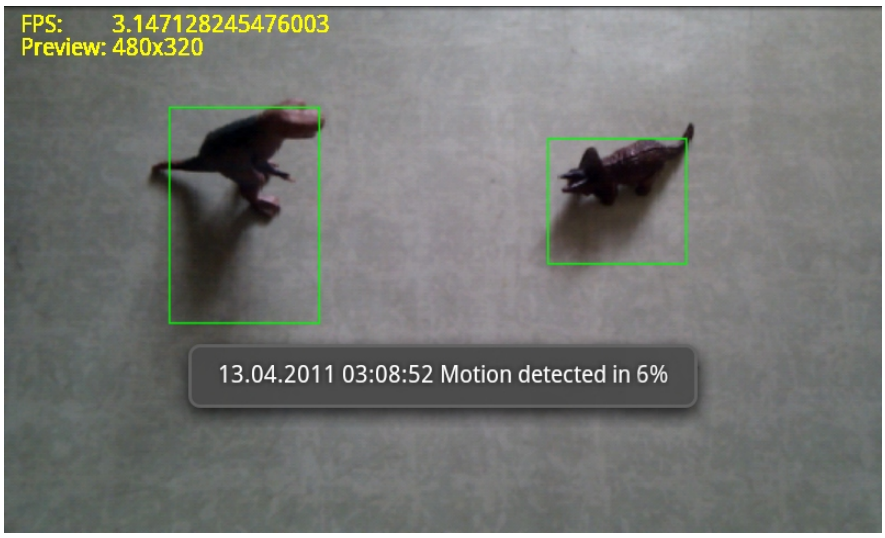
Obrázek 3.3 ukazuje vzhled některých částí uživatelské rozhraní. Zleva doprava to jsou: Hlavní menu, Nastavení, Volba ROI s vysunutou nabídkou menu (používá ikony ze systémových zdrojů pro zachování systémového vzhledu a úspoře prostředků). Výběr ROI se dělá posouváním jednotlivých rohů červeného obdélníku prstem.



Obrázek 3.3: Ukázka oken aplikace

Na obrázku 3.4 je pak k vidění obrazovka samotného detektoru v situaci, kdy byl dete-

kován dostatečný pohyb pro vyvolání upozornění ve formě Toast Notification[1]. Detekované pohybující se objekty detektor ohraničí zelenou čarou.



Obrázek 3.4: Demonstace výstupu detektoru při detekovaném pohybu

3.5 Vyhodnocení a testování

Tato podkapitola se bude zabývat vyhodnocením výkonu aplikace a testovacích telefonů za různých podmínek a testováním aplikace, a to jak ze strany funkčnosti, tak i robustnosti. Zaměřím se zde zejména na porovnání výkonu aplikace v Javě a v nativním kódu a také na rozdíly v rychlosti nativního zpracování na procesorech ARMv7-A s použitím jednotky NEON.

3.5.1 Výkon

Pro porovnání výkonu jsem navrhl sadu testů. Jednotlivé testy jsou více či méně závislé na celé aplikaci. Testy byly navrženy tak, aby demonstrovaly:

- Rozdíly výkonu testovacích zařízení.
- Rychlost zpracování v Javovském prostředí.
- Rychlost zpracování nativním kódem.
- Rychlost zpracování nativním kódem s využitím NEON jednotky procesoru ARMv7.

Výkonnostní testování probíhalo na čtyřech rozdílných telefonech. Pro porovnatelnost výsledků byly všechny početní operace prováděny se stejně velkými snímky v rozlišení 480x320 px. Výpis testovacích telefonů obsahuje tabulka 3.1.

[1]Toast Notification je informační zpráva, která se zobrazí na povrchu okna a po chvíli zase zmizí. Tato zpráva se velikostí přizpůsobí svému obsahu a není v interakci s uživatelem.

Název telefonu	CPU	ARM	RAM	Verze OS
LG P500 Optimus One	600 MHz (Qualcomm)	v6	512 MB	2.2
HTC Desire	1 Ghz (Qualcomm)	v7-A	576 MB	2.2
LG Optimus 2X	1 GHz Dual Core (Nvidia)	v7-A	512 MB	2.2
Google Nexus S	1 GHz (Samsung)	v7-A	512 MB	2.3

Tabulka 3.1: Seznam telefonů použitých pro výkonnostní testy

Test 1 - Snímky za sekundu (FPS)

První test porovnával rychlost celé aplikace na telefonech v různých konfiguracích. Výstupem testu byla hodnota FPS (snímky za sekundu), které dokázal detektor zpracovat. V tabulce 3.2 jsou vidět průměrné naměřené hodnoty při běhu aplikace na všech čtyřech telefonech za stejných okolních podmínek. Názvy sloupců značí použitou verzi nativní knihovny[2]. Z tabulky vyplývá, že rychlost běhu aplikace je dosti závislá na hardwarovém vybavení telefonu. Za povšimnutí stojí zejména rozdíl ve výkonnosti u telefonů HTC Desire a Google Nexus S. Přesto, že mají oba telefony stejný takt CPU, telefon Nexus S má téměř **o třicet procent lepší výsledky**. Hlavní příčinou je pravděpodobně novější a optimalizovanější verze OS Android a také procesor od jiného výrobce. Za povšimnutí také stojí porovnání mezi rychlostí aplikace s využitím a bez využití nativního kódu optimalizovaného pro procesory typu ARMv7-A. Jeho využití přináší **nárůst rychlosti až o jednu třetinu**.

	armeabi	armeabi-v7a
LG Optimus 2X	3,77 fps	4,93 fps
Google Nexus S	3,29 fps	4,51 fps
HTC Desire	2,75 fps	3,21 fps
LG P500 Optimus One	1,12 fps	x

Tabulka 3.2: Porovnání FPS

V levém horním rohu obrázku 3.4 je vidět jeden ze způsobů zobrazování naměřených hodnot výkonnostního testu na displeji.

Test 2 - Zpracování množství dat

Po změření rychlosti aplikace následovaly testy, které se snažily, s co nejvyšší mírou výpovědní hodnoty, porovnat rozdíly jak mezi rychlostí jednotlivých telefonů, tak zejména mezi rychlostí výpočtu prováděného na straně Javovského kódu, v nativním kódu a v nativním kódu zkompilovaném pro ARMv7-A. Testy byly dvou druhů:

- Konverze YUV formátu z kamery do RGB formátu a následné zobrazení
- Dávkový převod velké sady stejných snímků z formátu YUV do RGB formátu

V tabulkách 3.3, 3.4, 3.5 a 3.6 můžeme vidět výsledky stejného testu na různých konfiguracích. Můžeme vyčíst, že na nejpomalejším telefonu je rychlost při nativním zpracování

[2] **armeabi** je klasická verze podporovaná všemi ARM procesory pro Android, **armeabi-v7a** je knihovna přeložená pro instrukční sadu procesorů ARMv7-A.

Java	11,75 fps	13,26 fps	13,19 fps	13,20 fps	13,16 fps
C (armeabi)	11,66 fps	13,04 fps	13,13 fps	13,09 fps	13,15 fps
C (armeabi-v7a)	12,21 fps	13,66 fps	13,63 fps	13,67 fps	13,69 fps

Tabulka 3.3: Průměrné množství snímků (480x320 px) za sekundu v deseti sekundových intervalech, při konverzi YUV - RGB na telefonu Google Nexus S

Java	8,90 fps	10,10 fps	10,16 fps	10,11 fps	10,21 fps
C (armeabi)	9,08 fps	10,37 fps	10,38 fps	10,45 fps	10,44 fps
C (armeabi-v7a)	9,37 fps	10,72 fps	10,72 fps	70,73 fps	10,71 fps

Tabulka 3.4: Průměrné množství snímků (480x320 px) za sekundu v deseti sekundových intervalech, při konverzi YUV - RGB na telefonu LG Optimus 2X

Java	5,65 fps	5,65 fps	5,63 fps	5,64 fps	5,59 fps
C (armeabi)	5,80 fps	5,76 fps	5,78 fps	5,83 fps	5,81 fps
C (armeabi-v7a)	5,89 fps	5,86 fps	5,88 fps	5,90 fps	5,89 fps

Tabulka 3.5: Průměrné množství snímků (480x320 px) za sekundu v deseti sekundových intervalech, při konverzi YUV - RGB na telefonu HTC Desire

Java	2,02 fps	2,04 fps	2,02 fps	1,95 fps	1,98 fps
C (armeabi)	4,46 fps	4,40 fps	4,48 fps	4,46 fps	4,43 fps

Tabulka 3.6: Průměrné množství snímků (480x320 px) za sekundu v desetisekundových intervalech, při konverzi YUV - RGB na telefonu LG P500 Optimus One

výrazně vyšší, než při zpracování v Javě. Zajímavé je, že velikost tohoto rozdílu je v porovnání s výsledky na silnějších telefonech větší. Pravděpodobně je tomu tak proto, že na slabším telefonu je výraznější spotřeba výpočetního výkonu procesoru běhovým prostředím pro Javu.

Za povšimnutí stojí, že z tohoto testu vyšel lépe telefon Google Nexus S, než dvoujádrový LG Optimus 2X. Na vině je nedostatečná podpora pro využití více jader současné verze operačního systému Android u telefonu firmy LG a lépe optimalizovaný operační systém telefonu Nexus S.

Tabulky 3.3, 3.4 a 3.5 nám dále naznačují, že opravdu záleží na tom, zda je výpočetně náročný kód napsán v jazyce Java nebo v jazyce C a zda je využito jednotky pro vektorové zpracování. Rozdíly zde však stále nejsou na první pohled velké. Je tomu tak proto, že tento test je stále dost zatížen okolními operacemi aplikace, jako je načítání obrázků z kamery, vykreslování snímků, atd. Proto jsem navrhl poslední test, kde jsem se pokusil tyto vlivy maximálně omezit.

Test 3 - Dávkové zpracování

Poslední test je opět převodem z formátu YUV do RGB, tentokrát je však převod proveden v dávce, v průběhu které není nic čteno z kamery ani vykreslováno na displej. Pracuje se s jedním obrázkem v YUV formátu (pole bytů) a jedním obrázkem pro formát RGB (pole typu integer), který je cyklicky přepisován.

	100 iterací	**1000 iterací**
Java	1,96 s	19,36 s
C (armeabi)	1,58 s	17,08 s
C (armeabi-v7a)	1,37 s	16,47 s

Tabulka 3.7: Dávkový převod obrázků z YUV do RGB na telefonu LG Optimus 2X

	100 iterací	**1000 iterací**
Java	2,24 s	22,07 s
C (armeabi)	2,35 s	23,69 s
C (armeabi-v7a)	1,95 s	20,25 s

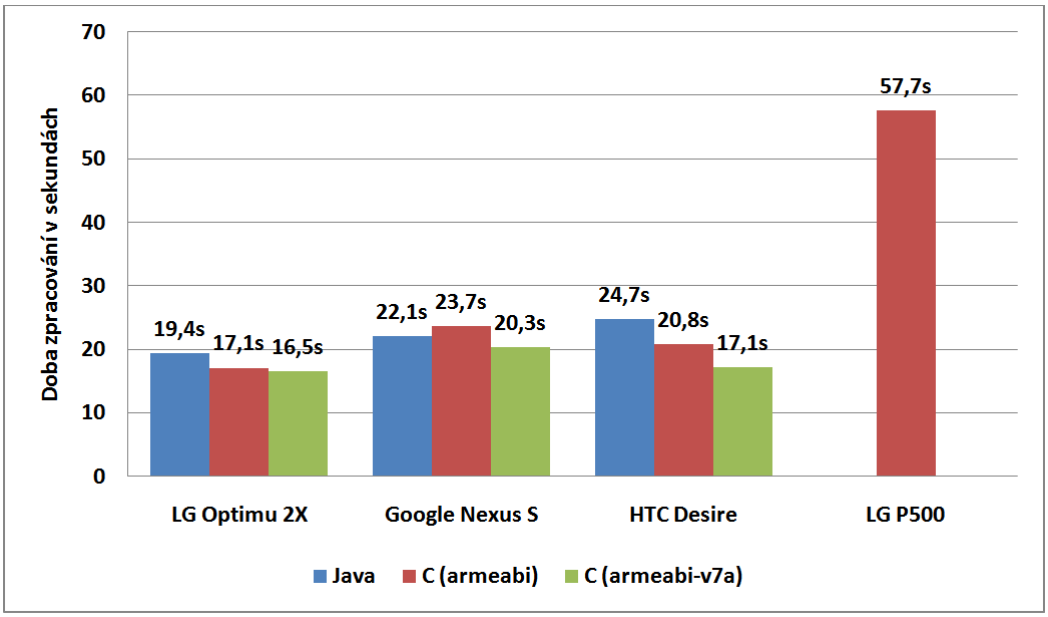
Tabulka 3.8: Dávkový převod obrázků z YUV do RGB na telefonu Google Nexus S

	100 iterací	**1000 iterací**
Java	2,42 s	24,71 s
C (armeabi)	2,11 s	20,83 s
C (armeabi-v7a)	1,69 s	17,10 s

Tabulka 3.9: Dávkový převod obrázků z YUV do RGB na telefonu HTC Desire

Tabulky 3.7, 3.8 a 3.9 nám ukazují, že využití instrukcí moderních procesorů ARMv7-A nám může, u výpočetně náročných operací, přinést **zrychlení o téměř 20%** (HTC Desire) v porovnání s běžnými ARM instrukcemi. Oproti kódu napsanému v Javě je toto **zrychlení**

přes 30%. Tyto rozdíly, společně s porovnáním se stejným testem na telefonu LG P500, jsou dobře patrné z grafu na obrázku 3.5[3]. Přesto se zdá, že s nárůstem výkonu telefonu tyto rozdíly klesají. Měření na telefonu Google Nexus S ukazuje zvláštní výchylku, kdy je výsledek výpočtu v Javě rychlejší, než v nativním kódu bez optimalizace pro ARMv7-A. Tato výchylka může být způsobena nepřesností měření nebo dobrou optimalizací OS Android 2.3.



Obrázek 3.5: Graf rychlosti dávkového zpracování 1000 snímků.

Tabulka 3.10 zobrazuje stejný test jako předchozí tabulky, nicméně na pomalejším mobilním zařízení. Rozdíl mezi zpracováním v Javě a v nativním kódu je zde propastný. Předpokládám, že je tomu tak proto, že slabší telefon má problém se vyrovnat se spuštěným běhovým prostředím jazyka Java a současným zpracováním velmi náročné operace.

	100 iterací	**1000 iterací**
Java	33,10 s	328,65 s
C (armeabi)	5,78 s	57,67 s

Tabulka 3.10: Dávkový převod obrázků z YUV do RGB na telefonu LG P500 Optimus One

3.5.2 Testování

Funkčnost

Funkčnost aplikace byla průběžně testována při vývoji. Aplikaci jsem testoval jak v telefonu, tak na emulátoru, který je součástí Android SDK. Klasické uživatelské testy byly zaměřeny zejména na použitelnost aplikace s cílem vytvořit stabilní aplikaci s intuitivním uživatelským rozhraním. Další testování bylo provedeno pro ověření robustnosti aplikace.

[3]Pro zachování názorného rozlišení je vypuštěna extrémní naměřená hodnota v Javě na telefonu LG P500, která je vidět v tabulce 3.10.

Robustnost

Aplikace běžící na systému Android mohou být v době běhu často přerušeny vnějšími vlivy a externími událostmi. Nejčastěji se jedná o přerušení příchozím hovorem nebo SMS zprávou. Vývojové prostředí Android SDK obsahuje nástroj **DDMS** (Dalvik Debug Monitor Server), který usnadňuje testování a debugging aplikací. Mimo jiné umožňuje pořizování snímků obrazovky telefonu, simulaci příchozího hovoru a simulaci příchozí SMS.

Při testu příchozího hovoru, v době spuštěné detekce, detektor na emulátoru po ukončení hovoru zamrzl. Na fyzickém zařízení však tento test proběhl v pořádku. Problém bych viděl ve špatné podpoře emulátoru pro video snímač.

Test SMS proběhl v pořádku.

Dalším zajímavým testovacím nástrojem je Application Exerciser Monkey pro testování uživatelského rozhraní aplikace. Jedná se o pseudo-náhodný generátor uživatelských akcí nad aplikací. Z příkazové řádky spustíme program např. příkazem:

$$\$\ adb\ shell\ monkey\ -p\ com.motiondetector\ -v\ 500$$

Přepínač $-p$ určuje balíček aplikace a přepínačem $-v$ nastavíme počet uživatelských akcí, které budou nad aplikací programem provedeny.

Kapitola 4

Závěr

Cílem této práce bylo seznámení se s moderní a rychle se rozvíjející platformou Android, dále pak s procesory (a jejich jednotkami na vektorové zpracování) používanými na zařízeních s touto platformou, seznámení se s algoritmy pro detekci pohybu ve video sekvenci a vytvoření demonstrační aplikace s využitím těchto znalostí.

Při vyhodnocení byl patrný rozdíl rychlosti při použití nativního kódu, zejména pak při jeho použití s využitím možností vektorového zpracování na procesorech typu ARMv7-A. Vzhledem k tomu, že se ale jednalo o uměle vytvořené podmínky s extrémně náročným výpočetním blokem a že u ostatních testů, které byly prováděny i s okolním důležitým kódem aplikace, nebyly tyto rozdíly tak velké. Potvrdil se tak předpoklad a doporučení tvůrců systému Android, a sice že nativní kód je vhodné využívat pouze v opodstatněných případech a výkonnostně kritických sekcích. Navíc je vidět, že u výkonnějších telefonů jsou rozdíly podstatně menší, než u telefonu s pomalejším procesorem.

Co se samotné aplikace týče, ukázalo se, že použitý algoritmus detekce pohybu je velmi citlivý na změny prostředí a je proto nutné jej kalibrovat pro různá prostředí, ve kterých má být využit. Tyto problémy jsou ještě umocněny nepříliš kvalitními video snímači mobilních telefonů, které mají většinou problémy vyrovnat se s obtížnými světelnými podmínkami a mají tak často nekvalitní výstup doplněný o množství šumových informací.

Díky dostupným vývojovým nástrojům a vzorovým částem kódu na internetu se při vývoji nevyskytlo mnoho velkých problémů. Největším problémem, který se také značnou měrou projevil na výkonnosti aplikace, byla absence možnosti přistoupit ke kameře zařízení z nativního prostředí a bylo tak nutné implementovat přeposílání jednotlivých snímků z Javovského prostředí.

Vzniklá aplikace je zajímavá tím, že jako jedna z mála využívá ve velké míře možnosti vývoje složitějších a náročných algoritmů v nativním kódu.

4.1 Pokračování vývoje a možná vylepšení

Již teď je zřejmé, že by aplikace mohla do budoucna obsahovat celou řadu rozšíření i vylepšení. Bylo by možné zvýšit výkon různými optimalizacemi algoritmu detektoru pohybu, například zmenšením velikosti snímku nad kterým jsou prováděny operace, počítáním pouze se sudými/lichými řádky/sloupci, atd. Další vylepšení výkonu by mohli přinést optimalizace nativního kódu použitím konkrétních instrukcí instrukční sady pro procesory jednotku NEON.

Co se týče výsledků samotného detektoru, i zde se nabízí spousta možností budou-

cího vývoje. Bylo by například možné zlepšit algoritmus detekce (na úkor zvýšení nároků aplikace). Nedostatkem mnou použitého algoritmu je jeho požadavek na statické umístění kamery, to znamená, že nemá schopnost vyrovnat se s pohybem video snímače. Tento nedostatek by se také dal odstranit nebo redukovat přidáním nějakého algoritmu detekujícího pohyb scény. To by ovšem opět zvýšilo požadavky na systém.

Z vylepšení aplikace, které by mohl uvítat uživatel, by bylo zajímavé například přidat možnost nahrávání video sekvence po dobu detekovaného pohybu, případně automaticky odesílat fotografie či video e-mailem nebo na webový server. Bylo by zde určitě vhodné udělat průzkum a zjistit tak, co by uživatelé aplikace ocenili.

Literatura

[1] Android developers. 2011, [online], [cit. 2011-03-26].
URL http://developer.android.com

[2] ARM architecture. 2011, [online], [cit. 2011-03-30].
URL http://en.wikipedia.org/wiki/ARM_architecture

[3] ARM The Architecture for the Digital World. 2011, [online], [cit. 2011-03-30].
URL http://arm.com

[4] FOURCC - YUV to RGB Conversion. 2011, [online], [cit. 2011-03-18].
URL http://www.fourcc.org/fccyvrgb.php

[5] Ketai: Android-based Processing framework for collecting device sensor & image data. 2011, [online], [cit. 2011-02-20].
URL http://ketai.googlecode.com/svn/trunk/ketai/src/edu/uic/ketai/inputService/KetaiCamera.java

[6] BRADSKI, G.; KAEHLER, A.: *Learning OpenCV*. O'Reilly & Associates, Inc., první vydání, 2008, ISBN 978-0-596-51613-0.

[7] CHADIM, P.: *Detekce pohybu ve video sekvenci*. Diplomová práce, Vysoké učení technické v Brně, Fakulta informačních technologií, 2008.

[8] DARCEY, L.; CONDER, S.: *Sams Teach Yourself Android^{TM} Application Development in 24 Hours*. SAMS, první vydání, 2010, ISBN 978-0-321-67335-0.

[9] HEIKKILÄ, M.; PIETIKÄINEN, M.: A Texture-Based Method for Modeling the Background and Detecting Moving Objects. 2005, [online].
URL http://www.ee.oulu.fi/mvg/publications/show_pdf.php?ID=662

[10] KATYSOVAS, T.: A first look at Google Android. 2007-2008, [online].
URL http://www.kandroid.org/s2/doc/android.pdf

[11] Kolektiv autorů: OpenCV. [online], [cit. 2011-03-18].
URL http://opencv.willowgarage.com/wiki/

[12] Kolektiv autorů: RGB Color Model. [online], [cit. 2011-03-22].
URL http://en.wikipedia.org/wiki/RGB_color_model

[13] MURRAY, J. D.; van RYPER, W.: *Encyclopedia of Graphics File Formats*. O'Reilly & Associates, Inc., druhé vydání, 1996, ISBN 1-56592-161-5.

[14] MUSIL, J.: *Detekce pohybujících se objektů ve video sekvenci*. Diplomová práce, Vysoké učení technické v Brně, Fakulta informačních technologií, 2008.

[15] SMEJKAL, J.: *3D aplikace pro platformu Android*. Diplomová práce, Vysoké učení technické v Brně, Fakulta informačních technologií, 2010.

[16] TOMEC, M.: *Optimalizace rozpoznávání řeči pro mobilní zařízení*. Diplomová práce, Vysoké učení technické v Brně, Fakulta informačních technologií, 2010.

[17] WANG, Z.: OpenCV in Android. 2010, [online], [cit. 2011-02-20].
URL http://www.stanford.edu/~zxwang/android_opencv.html

Příloha A

Obsah DVD

- **bin** - spustitelný .apk balíček aplikace
- **doc** - soubory pro vytvoření textové části bakalářské práce
- **src** - projekt pro vývojové prostředí Eclipse se zdrojovými soubory aplikace, README.TXT s instrukcemi pro překlad aplikace
- **poster** - plakát
- **video** - prezentační video