VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ
FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

ÚSTAV BIOMEDICÍNSKÉHO INŽENÝRSTVÍ
DEPARTMENT OF BIOMEDICAL ENGINEERING

STANOVENÍ KREVNÍHO TLAKU POMOCÍ CHYTRÉHO TELEFONU
BLOOD PRESSURE ESTIMATION USING SMARTPHONE

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE — **Bc. Martin Vařečka**
AUTHOR

VEDOUCÍ PRÁCE — **Ing. Andrea Němcová**
SUPERVISOR

BRNO 2018

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

Diplomová práce

magisterský navazující studijní obor **Biomedicínské inženýrství a bioinformatika**
Ústav biomedicínského inženýrství

Student: Bc. Martin Vařečka ***ID:*** 164219
Ročník: 2 ***Akademický rok:*** 2017/18

NÁZEV TÉMATU:

Stanovení krevního tlaku pomocí chytrého telefonu

POKYNY PRO VYPRACOVÁNÍ:

1) Prostudujte možnosti měření tlaku krve (TK), zaměřte se zejména na metody využívající chytrý telefon. Seznamte se s prostředím pro vývoj aplikací pro Android (alternativně iOS). 2) Nasnímejte testovací data pomocí vhodných senzorů chytrého telefonu a referenční data pomocí některé z konvenčních metod pro měření TK. 3) Vytvořte algoritmus pro stanovení TK z nasnímaných dat. 4) Optimalizujte algoritmus. Statisticky porovnejte výsledky s referencí. 5) Vytvořte aplikaci pro chytrý telefon a popište její architekturu. 6) Diskutujte výsledky, použitelnost aplikace v reálném životě, vliv prostředí, případně vliv použitého chytrého telefonu.

DOPORUČENÁ LITERATURA:

[1] CHANDRASEKARAN, V., R. DANTU, S. JONNADA, S. THIYAGARAJA a K. P. SUBBU. Cuffless Differential Blood Pressure Estimation Using Smart Phones. IEEE Transactions on Biomedical Engineering. 2013, 60(4), 1080-1089. ISSN 0018-9294.

[2] DIAS JUNIOR, A., S. MURALI, F. RINCON a D. ATIENZA. Methods for reliable estimation of pulse transit time and blood pressure variations using smartphone sensors. Microprocessors and Microsystems. 2016, 46, 84-95. ISSN 01419331.

Termín zadání: 5.2.2018 ***Termín odevzdání:*** 18.5.2018

Vedoucí práce: Ing. Andrea Němcová
Konzultant:

prof. Ing. Ivo Provazník, Ph.D.
předseda oborové rady

UPOZORNĚNÍ:
Autor diplomové práce nesmí při vytváření diplomové práce porušit autorská práva třetích osob, zejména nesmí zasahovat nedovoleným způsobem do cizích autorských práv osobnostních a musí si být plně vědom následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., včetně možných trestněprávních důsledků vyplývajících z ustanovení části druhé, hlavy VI. díl 4 Trestního zákoníku č.40/2009 Sb.

ABSTRAKT

Krevní tlak je jedním ze základních ukazatelů zdravotního stavu kardiovaskulárního systému. Vysoký krevní tlak je hlavním rizikovým faktorem pro vznik ischemické choroby srdeční, aterosklerózy a mozkových příhod. Z toho důvodu je důležité sledovat dlouhotrvající změny hodnoty krevního tlaku a včas na tyto změnu reagovat. Zařízení pro měření krevního tlaku není standardním vybavením domácnosti, zatím co velmi dobře vybavený chytrý telefon ano. Chytré telefony obsahují velké množství senzorů pomocí niž lze měřit biomedicínské signály. Tato práce se zaměřuje na vytvoření aplikace schopné stanovit hodnotu krevního tlaku pomocí dat získaných ze senzorů chytrého telefonu.

KLÍČOVÁ SLOVA

krevní tlak, TK, fotopletysmografie, PPG, fonokardiografie, FKG, doba přenosu pulzu, PTT, chytrý telefon, Android, Android Studio, Matlab, zpracování signálu

ABSTRACT

Blood pressure is one of the basic indicators of the health state of the cardiovascular system. High blood pressure is the main risk factor of ischemic heart disease, atherosclerosis and stroke. Therefore, it is important to monitor long-term changes in blood pressure and respond in time to these changes. Blood pressure meters are not standard household equipment, while a well-equipped smartphone is. Smartphones contain a large number of sensors capable of measuring biomedical signals. This thesis focuses on creating an application capable of determining blood pressure using data obtained from these sensors.

KEYWORDS

blood pressure, BP, photoplethysmography, PPG, phonocardiography, PCG, pulse transit time, PTT, smartphone, Android, Android Studio, Matlab, signal processing

VAŘEČKA, Martin. *Stanovení krevního tlaku pomocí chytrého telefonu*. Brno, 2018, 80 s. Diplomová práce. Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav biomedicínského inženýrství. Vedoucí práce: Ing. Andrea Němcová

Vysázeno pomocí balíčku `thesis` verze 3.03; `http://latex.feec.vutbr.cz`

PROHLÁŠENÍ

Prohlašuji, že svou diplomovou práci na téma „Stanovení krevního tlaku pomocí chytrého telefonu“ jsem vypracoval samostatně pod vedením vedoucího diplomové práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury na konci práce.

Jako autor uvedené diplomové práce dále prohlašuji, že v souvislosti s vytvořením této diplomové práce jsem neporušil autorská práva třetích osob, zejména jsem nezasáhl nedovoleným způsobem do cizích autorských práv osobnostních a/nebo majetkových a jsem si plně vědom následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon), ve znění pozdějších předpisů, včetně možných trestněprávních důsledků vyplývajících z ustanovení části druhé, hlavy VI. díl 4 Trestního zákoníku č. 40/2009 Sb.

Brno ....

podpis autora

PODĚKOVÁNÍ

Rád bych poděkoval vedoucí semestrální práce paní Ing. Andree Němcové za odborné vedení, konzultace, trpělivost a podnětné návrhy k práci.

Brno ....

podpis autora

Obsah

Seznam obrázků

Seznam tabulek

Seznam výpisů

Úvod

Vysoký krevní tlak, neboli hypertenze, je kardiovaskulární onemocnění, jenž je příčinou či společným faktorem kardiovaskulárních chorob. Tímto onemocněním trpí ve věku mezi 25 a 65 lety 35 % populace. Důvodem je fakt, že hypertenze v počátečním stádiu nemá téměř žádné příznaky, a proto většina postižených jedinců tento problém neřeší a nebo o něm vůbec neví. Cílem této práce je poskytnout těmto lidem možnost měření krevního tlaku bez vynaložení nákladů na pořízení zařízení, které je k tomuto účelu přímo určené.[46]

Chytré telefony jsou v dnešní době vybaveny výkonnými procesory, fotoaparáty s vysokým rozlišením, senzory zaznamenávající akcelerometrická data, světelnými senzory a kvalitními mikrofony. Na základě tohoto vybavení je možné snímat, zpracovávat a vyhodnocovat biomedicínská data.

Stanovit hodnotu krevního tlaku lze za pomoci fotopletysmografie a fonokardiografie. Fotopletysmogram lze nahradit fotoaparátem a přisvětlovací diodou. Fonokardiogram lze získat za pomoci mikrofonu telefonu přiloženého k hrudi. V obou signálech je pak sledován průchod pulsní vlny. Na základě průchodu pulsní vlny arteriálním řečištěm lze zjistit hodnotu krevního tlaku.

V první kapitole této práce je popsán krevní tlak a jeho vlastnosti. Také je zde popsáno fungování zařízení a principy metod měřící krevní tlak. Druhá kapitola popisuje operační systém Android a vývoj aplikací na tuto platformu. Třetí kapitola popisuje metody pro měření hodnoty krevního tlaku pomocí senzorů chytrého telefonu, výpočet hodnoty krevního tlaku a volbu zařízení vyhovující pro tento konkrétní účel. Ve čtvrté kapitole je popsán vývoj aplikace na operační systém *Android* umožňující akvizici dat. Pátá kapitola popisuje zpracování naměřených záznamů a stanovení hodnoty krevního tlaku.

1 Krevní tlak

Krevní tlak je jeden ze základních parametrů, které jsou důležité pro diagnostiku, řízení léčby, a hodnocení zdravotního stavu kardiovaskulárního systému. Z toho důvodu je měření krevního tlaku jedním z nejčastěji prováděných výkonů při návštěvě lékaře. Krevní tlak odráží stav psychické i fyzické zátěže, proto během dne jeho hodnota výrazně kolísá.[46]

1.1 Tlak krve

Oběhovou soustavu člověka lze chápat jako uzavřený hydrodynamický systém tvořený srdcem, cévami a krví. V tomto systému zastupuje srdeční sval svými pravidelnými stahy funkci pumpy zajišťující proudění krve elastickým potrubím tvořeným cévami. Výsledkem spolupráce srdeční aktivity a periferního odporu cév vzniká krevní tlak definovaný jako síla působící na stěnu cév, kterými proudí krev. Krevní tlak patří mezi základní parametry udávající hemodynamický stav oběhové soustavy. V medicíně se krevní tlak udává v jednotkách *mmHg* (milimetr rtuťového sloupce) resp. *torr*. Vztah mezi obecně používanými jednotkami tlaku a jednotkami používanými v medicíně je uveden v rovnici 1.1.[41]

$$1\ atm = 101325\ Pa = 760\ torr = 760\ mmHg \tag{1.1}$$

Tlak se v různých částech krevního řečiště liší, a proto je krevním tlakem obvykle míněn tlak v arteriálním řečišti. Dále se krevní tlak mění v průběhu srdečního cyklu. Srdeční cyklus se skládá z fáze relaxace zvané **diastola** a fáze kontrakce zvané **systola**. Krevní tlak kolísá mezi dvěma extrémy. Nejvyšší hodnoty je dosaženo při systole a nejnižší při diastole.[34, 41]

Systolický tlak je nejvyšší tlak ve vypuzovací fázi srdečního cyklu. V této fázi je překonán tlak v aortě, což způsobí otevření poloměsíčitých chlopní a následné vypuzení krve do arteriálního řečiště. Na konci systoly začne tlak v komorách klesat. Díky vyššímu tlaku v arteriálním řečišti jsou uzavřeny poloměsíčité chlopně a nastává tak fáze plnění komor. Komory se opět plní krví, čímž klesá tlak v arteriálním řečišti. Diastolický tlak je pak nejnižší tlak ve fázi plnění komor.[29, 50, 56]

Krom hodnot systolického (p_S) a diastolického tlaku (p_D) se dále používá hodnota středního (p_M) a pulsního tlaku (p_P). Pulsní tlak lze vypočítat dle vztahu 1.2. Střední tlak lze vypočítat dle vztahu 1.3.[41]

$$p_P = p_S - p_D \tag{1.2}$$

$$p_M = p_D + \frac{p_P}{3} \tag{1.3}$$

Systolický tlak se v průběhu života postupně zvyšuje. U kojenců a malých dětí se pohybuje v rozmezí 80–85 mmHg. U větších dětí v rozmezí 110–120 mmHg. U dospělého jedince se systolický tlak pohybuje v rozsahu 100–140 mmHg, přičemž tlak 120 mmHg je brán jako fyziologická hodnota. Diastolický tlak je měřítkem periferního odporu a nepodléhá výraznějším změnám, jako tlak systolický. Krom prvních let života, kdy se pohybuje v rozsahu 45–50 mmHg, se diastolický tlak pohybuje kolem hodnoty 80 mmHg. Hodnoty krevního tlaku v různých obdobích života jsou uvedeny v tab. 1.1.[42]

Tab. 1.1: Krevní tlak dle věku [42]

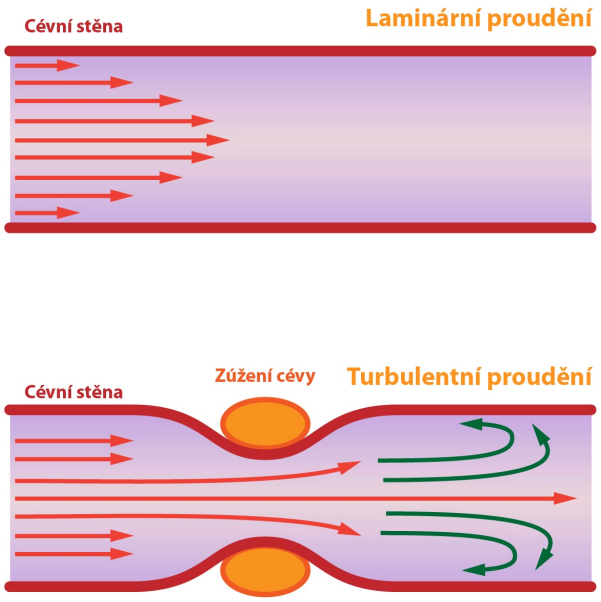
	Normální TK [mmHg]	Nízký TK [mmHg]	Vysoký TK [mmHg]
Kojenec	80/45	$<$ 80/45	$>$ 85/50
Větší dítě	110/70	$<$ 110/70	$>$ 120/80
Dospělý muž	120/80	$<$ 100/60	$>$ 140/90
Dospělá žena	120/80	$<$ 100/70	$>$ 140/90

Krev vypuzená srdcem proudí v cévách za fyziologických podmínek laminárně. Laminární proudění je charakteristické rovnoběžným pohybem jednotlivých vrstev, přičemž se jednotlivé vrstvy pohybují různou rychlostí. Vlivem nízkého tření se krev ve středu cévy pohybuje nejrychleji a naopak vlivem vysokého tření u stěny nejpomaleji. Za určitých podmínek vznikají v krevním proudu víry a proudění se mění z laminárního na turbulentní. Turbulentní proudění je charakteristické chaotickým promícháváním vrstev. Vznik vírů při turbulentním proudění je doprovázen zvukovými fenomény, kterých se využívá u většiny metod pro měření krevního tlaku.[50, 56]

Přechod z laminárního proudění na turbulentní předpovídá parametr Reynoldsova čísla, který je vypočítán dle vztahu 1.4. [29]

$$R_e = \frac{d \cdot v \cdot \rho}{\eta} \tag{1.4}$$

Kde, ve vztahu pro proudění krve cévami, je d průměr cévy, v lineární rychlost proudu krve, ρ hustota krve a η je dynamická viskozita krve. Reynoldsovo číslo je bezrozměrné. Je-li hodnota Reynoldsova čísla větší než 200, pak se v proudu objevují ojedinělé turbulence. Při překročení hodnoty 1 000 je již proudění plně turbulentní. Změna laminárního proudění v turbulentní je vyobrazena na obrázku 1.1. [29, 56]



Obr. 1.1: Změna laminárního proudění v turbulentní [29]

1.2 Konvenční měření krevního tlaku

Konvenční metody pro měření krevního tlaku lze rozdělit dle zásahu do organismu na invazivní (přímé) a neinvazivní (nepřímé). Při invazivním měření je nutný přímý kontakt s krevním řečištěm. Toho je dosaženo narušením cévní stěny pomocí jehly či katetru. Díky tomu je možné měřit tlak v periferních částech cévního řečiště. Přímé metody jsou vhodné pro měření dynamických změn krevního tlaku a získání velmi přesných hodnot. Naopak neinvazivní metody získávají data z povrchu těla. Obvyklou součástí nepřímých metod je použití tlakové manžety omezující průtok krve v měřeném místě, což způsobí zúžení cévy a vznik turbulentního proudění.[41]

Jelikož se praktická část této práce zaměřuje na zhotovení neinvazivní metody měření krevního tlaku pomocí mobilního telefonu, nejsou v této práci rozebrány metody invazivní.

Neinvazivní měření krevního tlaku je dnes nejpoužívanější metodou pro jeho stanovení. Hlavním důvodem je šetrnost vůči pacientovi a časová nenáročnost. Většina neinvazivních metod je nespojitá.[41]

1.2.1 Palpační metoda

Jedná se o velmi jednoduchou manuální metodu, pro jejíž provedení je nutný pouze tonometr. Manžeta je umístěna nad loket ruky pacienta. Na zápěstí téže ruky je

nahmatán puls na vřetenní tepně (*arteria radialis*). Nafouknutím manžety nad hodnotu systolického tlaku dojde k uzavření pažní tepny (*arteria brachialis*), čímž vymizí pulsace na tepně vřetenní. Postupným upouštěním manžety dojde k obnovení pulsace. Hodnota odečtená ze stupnice tonometru při obnovení pulsace pak odpovídá hodnotě systolického tlaku. Nevýhodou této metody je nemožnost stanovení hodnoty tlaku diastolického.[27]

1.2.2 Auskultační metoda

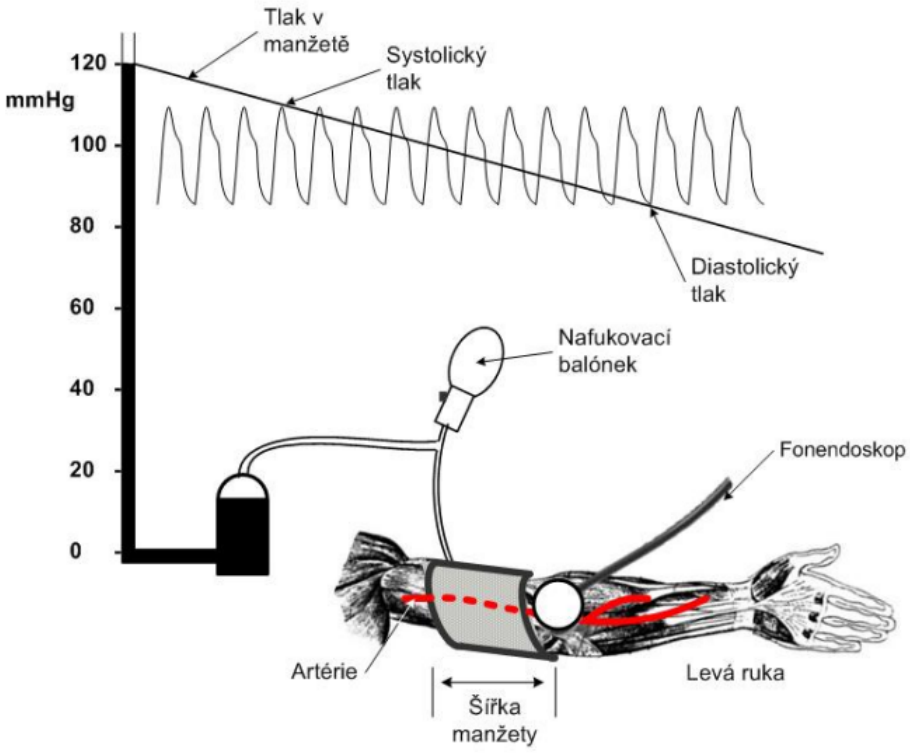
Princip této metody je v základu podobný metodě palpační. Rozdílem je detekce akustických fenoménů za pomoci fonendoskopu místo sledování pulsace. Hlavice fonendoskopu je zasunuta pod tlakovou manžetu, či je přiložena distálně nad arterii. Před nafouknutím manžety je nahmatán puls na vřetenní tepně. Tlaková manžeta je nafukována do vymizení pulsace. Tlak z manžety je postupně odpouštěn (rychlostí 2–3 mmHg/s). První slyšitelné zvuky jsou známkou průtoku krve arterií, kterým odpovídá hodnota systolického tlaku. Manžeta je dále upouštěna, přičemž se akustické fenomény zesilují. Po dosažení určitého maxima se začne hlasitost fenoménů zeslabovat, až úplně zaniknou. Tato hodnota odpovídá tlaku diastolickému.[34, 41, 46]

Výše zmíněné akustické fenomény byly objeveny v roce 1905 ruským chirurgem Nikolajem Sergejevičem Korotkovem – *Korotkovovy fenomény*. Za vznik těchto fenoménů je považována změna laminárního proudění na turbulentní a také naopak turbulentního v laminární v důsledků zúžení místa arterie natlakovanou manžetou. Korotkovy fenomény je možné rozdělit do 5 navazujících fází.[46]

I. První jasný tón připomínající kapající kohoutek (zároveň se objevuje pulsace).
II. Tóny mají charakter šelestu (delší a tlumenější).
III. Hlasitost tónů dosahuje maxima (ostré a hlasité).
IV. Dochází k oslabení tónů (méně zřetelné, měkké a tlumené).
V. Vymizení tónů.

Systolický tlak pak odpovídá hodnotě na stupnici tonometru při první fázi Korotkovových fenoménů a diastolický tlak při fázi páté. Princip auskultační metody je vyobrazen na obrázku 1.2.[41]

Auskultační metoda je nejspolehlivější a nejvyužívanější metodou pro stanovení hodnoty krevního tlaku. V oblasti kardiologie je považována za referenční. Metoda využívá minimum přístrojů a je méně technologicky náročná. I přesto jsou zde určitá omezení a zdroje chyb: obtížně se získávají hodnoty v hlučném prostředí, je velmi závislá na zručnosti osoby provádějící měření a také na volbě správného rozměru tlakové manžety.[34, 41, 46]



Obr. 1.2: Princip auskultační metody. [41]

1.2.3 Oscilometrická metoda

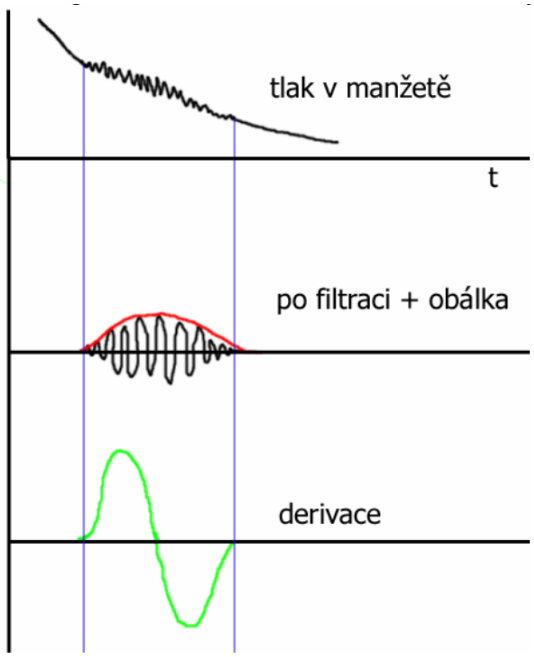
Metoda je založena na snímání tlakových změn uvnitř manžety, kterým odpovídají pulsace artérie. Velikost těchto pulsací je dána hodnotou transmurálního tlaku, který je dán rozdílem tlaku uvnitř a vně tepny. Transmurálního tlak lze vypočítat dle vztahu 1.5.[27]

$$p_{transmurální} = p_{krve} - (p_{tkání} + p_{manžety}) \tag{1.5}$$

Kde tlak uvnitř arterie (vnitřní) je dán přímo tlakem krve vypuzované srdcem. Tlak vně arterie (vnější) je dán působením okolních tkání (svaly, kůže apod.) a tlakem manžety. Tlak okolních tkání lze zanedbat vzhledem k podstatně většímu tlaku v manžetě. Ta je skrz okolní tkáň v těsném kontaktu s artérií. Společně tvoří mechanickou soustavu, přes kterou jsou přenášeny oscilace na tlakový senzor.[27]

Při zvýšení vnějšího tlaku (nafouknutí manžety) nad hodnotu tlaku vnitřního je artérie uzavřena. Postupným snižováním vnějšího tlaku dojde k opětovnému průchodu krve artérií a vzniku oscilací, jejichž amplituda postupně narůstá. Při určité hodnotě vnějšího tlaku dosahují oscilace maxima amplitudy. Maximum odpovídá střední hodnotě arteriálního tlaku. Po překročení této hodnoty amplituda oscilací klesá, až úplně vymizí.[27, 41]

Pro určení hodnot systolického a diastolického tlaku je průběh oscilací filtrován filtrem typu horní propust a následně je detekována obálka. Derivací obálky vznikají dva extrémy. Přenesením bodu s nejvyšší hodnotou na průběh tlaku v manžetě nalezneme hodnotu systolického tlaku. Diastolický tlak je pak nalezen přenesením bodu s nejnižší hodnotou na průběh tlaku v manžetě. Časový průběh signálu z tlakové manžety a jeho zpracování je vyobrazeno na obrázku 1.3.[27, 41]



Obr. 1.3: Časový průběh signálu z tlakové manžety a jeho zpracování.[41]

Tuto metodu dnes využívá řada dostupných tonometrů. Je vhodná i pro dlouhodobé monitorování, ale je velmi citlivá na pohyby pacienta v průběhu měření. Oproti auskultační metodě není závislá na okolním zvuku.[41]

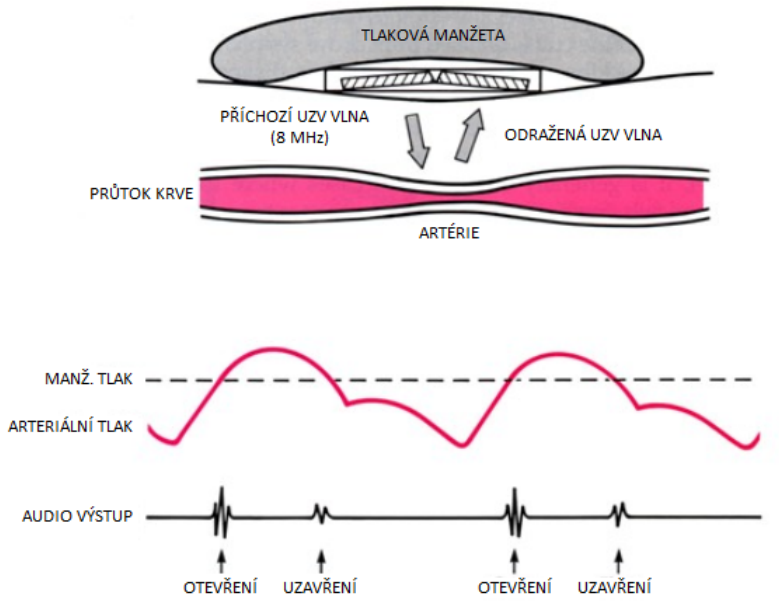
1.2.4 Ultrazvuková metoda

Ultrazvuková metoda využívá principu Dopplerova jevu. Stejně jako u předchozích metod je využito tlakové manžety, pod kterou je ovšem upevněno ultrazvukové vysílací a přijímací zařízení. Vyslaný ultrazvukový signál se dostává skrz tkáně až k artérii, kde se částečně odráží od pohybující se stěny artérie a krevních elementů (obzvláště červených krvinek). Dochází ke změně frekvence, která je zachycena přijímačem. Velikost změny je dána vztahem 1.6.[27]

$$f_D = \frac{2vf_0 \cdot cos(\alpha)}{c} \tag{1.6}$$

Kde v je rychlost průtoku krve, α je dopplerovský úhel, f_0 je frekvence vlnění vysílače a c je rychlost šíření ultrazvukové vlny ve tkáni. [27]

Při tlaku manžety v rozmezí systolického a diastolického tlaku jsou detekovány dva signály, které odpovídají otevírání a uzavírání artérie. Je-li tlak v manžetě vyšší než tlak systolický je artérie uzavřena, pokud je nižší, je artérie otevřena. Při zvyšování tlaku v manžetě se budou oba signály vzájemně přibližovat, až dojde k jejich splynutí. Toto splynutí pak odpovídá hodnotě systolického tlaku. Opačně je tomu při snižování tlaku v manžetě, kdy se oba signály od sebe vzájemně vzdalují, až dojde k vymizení jednoho z nich. Tento okamžik pak odpovídá hodnotě diastolického tlaku. Princip ultrazvukové metody je vyobrazen na obrázku 1.6.[27]



Obr. 1.4: Princip ultrazvukové (Dopplerovy) metody.[27]

1.2.5 Fotopletysmografická metoda

Tato metoda byla vyvinuta českým profesorem Janem Peňázem. Jedná se o způsob spojitého měření krevního tlaku. Na rozdíl od předchozích metod využívajících tlakové manžety, která je v tomto případě umístěna na prstu, je u této metody vždy zajištěn alespoň minimální průtok krve. Manžeta je navíc vybavena infračervenou diodou o vlnové délce 940 nm a fotočlánkem. Vlnová délka 940 nm je velmi dobře absorbována oxyhemoglobinem, který je obsažen v okysličené krvi. Tlak v manžetě je řízen zpětnovazebním regulačním systémem, tak aby se neměnila intenzita světla

a objem krve v prstu zůstal konstantní. Pulzace manžety jsou sledovány a vyhodnocovány fotopletysmografickou soustavou. Metoda je obvykle využívána u pacientů, kteří jsou v klidu, jelikož je velmi citlivá na pohyb.[41]

2 Operační systém Android

Tato kapitola pojednává o vývoji aplikací běžících na platformě Android. V následujících sekcích jsou shrnuty základní poznatky o operačním systému Android a jeho architektuře. Dále je popsáno vývojové prostředí *Android Studio* včetně nástrojů potřebných pro vývoj aplikací. Kapitola objasňuje pojmy, které jsou důležité pro následný vývoj aplikace.

2.1 Platforma Android

V současné době je Android nejrychleji rostoucí, nejrozšířenější a otevřený (open source) mobilní operační systém. Je založený na linuxovém jádře. O vývoj tohoto operačních systému se v současné době stará společnost Google. Android je primárně určen pro mobilní zařízení, jako jsou chytré telefony, tablety, nositelnosti (hodinky, sport testery) a jiné.

Původní společnost Android Inc. byla založena čtveřicí vývojářů v roce 2003. Původním cílem bylo navrhnout a vyvinout operační systém pro digitální fotoaparáty. Ovšem v té době nebyl tento trh příliš rozšířený, proto započali vývoj operačního systému pro mobilní zařízení.

V roce 2005 společnost Android Inc. odkoupila společnost Google, jehož cílem bylo prorazit na trh s mobilními zařízeními. V roce 2007 Google založil konsorcium OHA (Open Handset Alliance), pod které spadají výrobci mobilních telefonů, telekomunikační operátoři a technologické firmy, jejichž cílem bylo vytvořit nový systém pro mobilní zařízení. *Android Alpha* byl název první verze rozšiřitelného a flexibilního operačního systému, založeného na linuxovém jádře. Aplikace běžící na této platformě byly napsané v programovacím jazyce *Java*. Při vývoji byly kladeny požadavky, aby bylo možné systém integrovat do současných, ale i nových mobilních telefonů. Od roku 2007 prošel Android velkým množstvím změn, které zdokonalily tento operační systém.[4, 39]

2.2 Verze operačního systému

Operační systém Android se neustále vyvíjí a je rozšiřován o nové funkce. Vyžití tohoto operačního systému je v mobilních zařízeních čím dál častější. Hlavním důvodem je neustále se zvětšující obchod s aplikacemi, zvaný *Google Play*, a velmi příjemné uživatelské prostředí.

Využití jednotlivých verzí operačního systému Android je uvedeno v tab.2.1. Data byla naposledy aktualizována 9. 11. 2017. Z tabulky je patrné, že starších verzí postupně ubývá. V tabulce nejsou zobrazeny verze systému se současným využitím

menším než 0,1 %. Starší verze postupně ztrácejí podporu, jsou pomalé a nemají velké množství dostupných aplikací, jako verze nové. Aktuálně nejpoužívanější verze operačního systému Android nese název *Marshmallow* a verzové označení *6.0*.[24]

Tab. 2.1: Verze operačního systému Android [24]

Verze	Označení	API	Využití
2.3.3 - 2.3.7	Gingerbread	10	0.5 %
4.0.3 - 4.0.4	Ice Cream Sandwich	15	0.5 %
4.1.x	Jelly Bean	16	2.2 %
4.2.x		17	3.1 %
4.3		18	0.9 %
4.4	KitKat	19	13.8 %
5.0	Lollipop	21	6.4 %
5.1		22	20.8 %
6.0	Marshmallow	23	30.9 %
7.0	Nougat	24	17.6 %
7.1		25	3.0 %
8.0	Oreo	26	0.3 %

2.3 Architektura operačního systému

Pro vývoj aplikací je nutné seznámit se se základními informacemi o architektuře operačního systému a jeho principech. Operační systém Android se skládá z 5 hlavních vrstev, kterými jsou Linux Kernel, Libraries, Android Runtime, Application Framework a Applications.[43]

Linux Kernel

Nejnižší vrstvou operačního systému Android je upravené linuxové jádro. Původní jádro populárního operačního systému Linux obsahovalo příliš mnoho funkcí, které na mobilních zařízeních nelze využít. Proto bylo nutné provést jejich redukci. Samotné jádro zajišťuje přímou komunikaci s hardwarem mobilního zařízení. Též zajišťuje správu paměti a procesů, základní síťovou vrstvu a různé hardwarové ovladače pro komunikaci s periférierniemi.[43]

Jelikož Android může běžet na spoustě různých zařízení (hardwarově se lišících), bylo nutné na úrovni jádra vytvořit vrstvu HAL (Hardware Abstraction Layer), jenž vytváří rozhraní pro komunikaci vyšších vrstev systému s hardwarem. Díky tomu

není nutné, aby vývojář znal přesnou hardwarovou specifikaci zařízení, pro kterou aplikaci vyvíjí.[43]

Libraries

Nad jádrem se nachází vrstva nativních knihoven napsaných v jazyce *C* či *C++*. Ty tvoří mezivrstvu mezi linuxovým jádrem a vyššími vrstvami. Aplikace, které pracují s daty, zde mohou například využít relační databázový systém *SQLite*. Taktéž jsou zde grafické knihovny, knihovny médií a spousty dalších. Tato vrstva nahrazuje funkcionalitu, o kterou bylo jádro redukováno.[43]

Android Runtime

ATR (Android Runtime) je virtuální stroj, který vytváří běhové prostředí pro spuštěné aplikace napsané v jazyce *Java*. Každá aplikace v systému funguje jako samostatný proces, který využívá vlastní instanci virtuálního stroje.[43]

Application Framework

Další vrstva, ze které je operační systém Android sestaven je softwarová struktura poskytující opakovatelně použitelný software (ovládací prvky, ikony), přístup k službám a manažerům systému. Jedná se tedy o nejdůležitější vrstvu pro vývojáře.[43]

Applications

Nejvyšší vrstvu architektury operačního systému Android tvoří aplikace samotné. Obecně můžeme aplikace rozdělit na dva druhy. Prvním z nich jsou systémové aplikace, které slouží na telefonování, správu kontaktů, přijímání zpráv, fotografování a podobně. Druhým druhem jsou aplikace uživatelem doinstalované. Jedná se o aplikace stažené z obchodu *Google Play* či manuálně nainstalované.[43]

2.4 Vývoj aplikací pro platformu Android

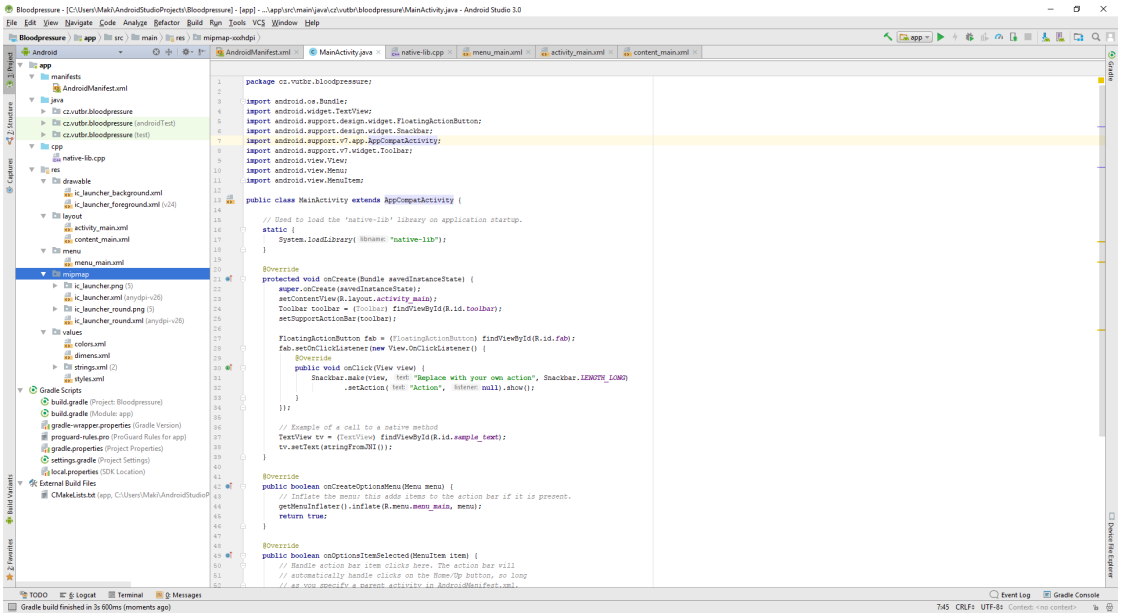
Aplikace pro platformu Android mohou být vyvíjeny v různých programovacích jazycích či jejich kombinacemi. Aktuálně je možné vyvíjet android aplikace v programovacím jazyce *Kotlin*, *Java* či *C++*.[8, 43]

2.4.1 Vývojové prostředí

Neoddělitelnou součástí vývoje je i volba vývojového prostředí. V současné době lze aplikace běžící na platformě Android vyvíjet v různých prostředích. Mezi nejvíce rozšířená vývojová prostředí patří *Android Studio* a *Eclipse*. Do nedávna bylo

oficiálním vývojovým prostředím *Eclipse*. Jedná se o flexibilní vývojové prostředí, které umožňuje nainstalovat doplňky pro další programovací jazyky, kterými jsou například *PHP*, *Python*, *Ruby*, *C++* a další. Právě díky flexibilitě a jednotnému prostředí je mezi vývojáři oblíbené. Pro vývoj v prostředí *Eclipse* je nutné stáhnout plugin ADT (Android Development Tools), který umožňuje pracovat s aplikacemi na platformě Android. Vzhledem k náročnosti konfigurace není toto vývojové prostředí začátečníkům doporučováno.[43]

Android Studio je oficiální vývojové prostředí založené na IntelliJ IDEA, určené primárně pro vývoj aplikací na platformu Android od společnosti Google. Je volně dostupné a multiplatformní. Instalace je velmi jednoduchá - není třeba doinstalovávat pluginy, na rozdíl od *Eclipse*. Součástí instalačního balíčku je Android Studio IDE (Integrated Development Environment), Android SDK Tools, kompilátor a základní emulátor. Vývojové prostředí po nainstalování, spuštění a založení projektu je vyobrazeno na obr. 2.1.[5, 43, 45]



Obr. 2.1: Vývojové prostředí Android Studio

2.4.2 Základy aplikace

Základními stavebními prvky jsou komponenty realizované jako třídy. Každá komponenta slouží jako vstupní bod skrz který může systém nebo uživatel vstoupit do aplikace. Existují čtyři různé druhy komponent, přičemž některé jsou závislé na ostatních.[8]

- Aktivity (Activities)
- Služby (Services)
- Posluchače (Broadcast receivers)
- Poskytovatelé obsahu (Content providers)

Každá z těchto komponent slouží k jinému účelu a má odlišný životní cyklus, který určuje jak je komponenta vytvořena a zničena.[8]

Aktivity

Aktivity jsou stavebními bloky uživatelského rozhraní. Jsou hierarchicky uspořádané. Po spuštění aplikace se tedy otevře hlavní spouštěcí aktivita, ze které jsou v případě potřeby spouštěny další aktivity. Jedna aktivita odpovídá právě jedné obrazovce, a proto aplikace obvykle obsahují více aktivit, mezi kterými se uživatel přepíná. Každá aktivita by měla mít svůj konkrétní účel. Tím je myšleno například vyplnění formuláře, nastavení parametrů či vybrání položky ze seznamu. Aktivita tedy definuje GUI (Graphical User Interface), přes které je zajištěna interakce uživatel-aplikace a opačně.[8]

Služby

Služby jsou komponenty, které běží na pozadí a provádí dlouhodobě běžící operace, jako je například přehrávání hudby či stahování. Také jsou využity k přístupu vzdálených zdrojů, u kterých není známa doba odezvy, například kontrola dostupnosti aktualizací či připojení k serveru. Na rozdíl od aktivit, služby neposkytují žádné uživatelské rozhraní.[1, 43]

Posluchače

Komponenta Broadcast receivers není do českého jazyka obvykle překládána. Poměrně volným překladem vzniká název *Přijímače vysílání* či *Posluchače*. Ovšem ani jeden z těchto překladů zcela nevystihuje vlastnost této komponenty. Komponenta primárně slouží k naslouchání oznámení, kterými mohou být systémové či vlastní zprávy.[8, 43]

Vše se odehrává na pozadí zařízení bez jakéhokoliv uživatelského rozhraní. Událost, kterou posluchače zachytávají jsou takzvané záměry (Intents). Vydavatelé (systém či aplikace) vytvářejí záměry, které jsou následně směřovány do vysílání. Ty jsou zachytávány přijímači, které mají stejné záměry objednané či registrované.[8, 43]

Ideálním příkladem je příjetí SMS zprávy. V momentě, kdy dorazí SMS do telefonu, operační systém Android to vhodně oznámí tím, že vyšle záměr *SMS_received*. Všechny přijímače, které mají stejné záměry tento záměr přijmou a spustí službu,

která stáhne příchozí SMS zprávu. Poté si uživatel zprávu skrz některou z aplikací zobrazí. Dalším dobrým příkladem je například upozornění na nízký stav baterie.[43]

Jak již bylo naznačeno výše, existují 2 druhy *broadcastů* a to systémové či vlastní.[8, 43]

Poskytovatelé obsahu

Poslední komponentou jsou poskytovatelé obsahu. Jak již název napovídá, umožňují ukládání a sdílení dat mezi více aplikacemi či procesy. Aplikace tak může přistoupit k datům jiné aplikace, které vystupují jako poskytovatelé obsahu.[1, 43]

3 Měření krevního tlaku pomocí telefonu

Mobilní telefony jsou v dnešní době neodmyslitelnou součástí našich životů. Z toho důvodu jsou tato zařízení neustále rozšiřována o nové funkce. Dnes se již prakticky nesetkáme s *klasickým* mobilním telefonem, jenž byl omezen pouze na volání a posílání zpráv. Tyto telefony byly vytlačeny telefony *chytrými* (smartphone), které jsou postaveny na pokročilých operačních systémech. Momentálně nejznámějšími operačními systémy jsou *Android*, *iOS* a *Windows Phone*. Operační systém umožňuje uživateli instalovat aplikace třetích stran a tím značně rozšířit funkcionalitu zařízení.

Krom programového vybavení (software - SW) je taktéž důležité i technické vybavení (hardware - HW) zařízení. Jednou z velkých skupin HW jsou senzory, pomocí nichž zařízení interaguje s okolím. Tyto senzory je možné využít pro snímání biomedicínských dat. Díky velmi vysokému výpočetnímu výkonu současných zařízení je možné tyto biomedicínské signály na tomtéž zařízení i zpracovat.

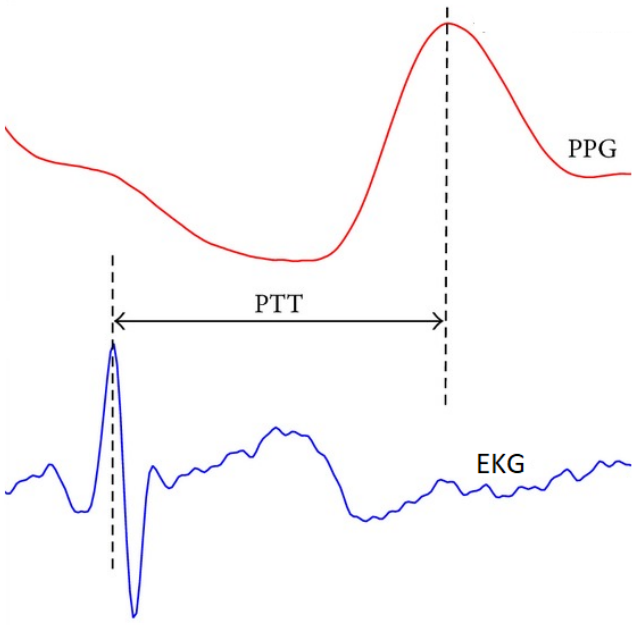
V posledních letech byly představeny nové způsoby měření krevního tlaku pomocí různých senzorů. Jedná se především o experimentální metody, které nejsou v běžné praxi prozatím využívány. Nejslibnější metody z této oblasti jsou založeny na měření doby přenosu pulsu (Pulse Transit Time - PTT), který udává čas, za který projde pulsní vlna mezi dvěma místy arteriálního řečiště.[26, 28, 31, 33]

3.1 Doba přenosu pulsu

Základní princip všech metod měřící PTT je fakt, že tuhost artérií zvyšuje hodnotu krevního tlaku, u kterého je předpokládána přímá korelace s rychlost šíření pulsní vlny (Pulse Wave Velocity - PWV) v arteriálním řečišti. PWV patří dnes mezi základní metodu hodnocení arteriální tuhosti – ztráty elasticity. Ztráta elasticity je běžná s přibývajícím věkem. To je způsobeno stárnutím a mechanickým namáháním elastinových vláken ve stěnách artérií. Míru tuhosti daného úseku arteriálního řečiště charakterizuje parametr PWV, který lze vypočítat dle vztahu 3.1, jenž odpovídá vzdálenosti mezi dvěma body arteriálního řečiště $\triangle x$ uražené pulsní vlnou za čas PTT. Rychlost šíření pulsní vlny je přímo úměrná tlaku krve. Při zvýšení krevního tlaku dojde ke zvýšení tuhosti artérií a tedy ke zvýšení PWV, což má za následek snížení PTT. Při snížení tlaku dojde ke snížení tuhosti artérií, snížení PWV a zvýšení PTT. Pomocí PTT je pak možné predikovat změny krevního tlaku během krátkých časových úseků.[28, 31]

$$PWV = \frac{\triangle x}{PTT} \tag{3.1}$$

Parametr PTT byl nejprve měřen mezi dvěma body konkrétní artérie. Poté byl první bod nahrazen měřením signálů ze srdečního svalu v podobě elektrokardiogramu (EKG), fonokardioframu (FKG) či impedančního kardiogramu (IKG).V případě EKG vlna R v odpovídá době otevření poloměsíčitých chlopní, vypuzení krve do aorty a vzniku pulsní vlny. V druhém bodě je poté detekována pulsní vlna pomocí tlakové manžety či fotopletysmografu umístěném na konečku prstu. Princip měření PTT je vyobrazen na obrázku 3.1. Pro detekci pulsní vlny bývá obvykle využit záznam EKG a fotopletysmograf (Photopletysmograph - PPG) umístěný na konečku prstu.[26, 31]



Obr. 3.1: Princip měření doby přenosu pulsu (PTT) z EKG a PPG.[31]

Bod v PPG signálu odpovídající příchodu pulsní vlny je v různých algoritmech jiný. Některé zdroje udávají 50 % maxima pulsní vlny [31], jiné zdroje maximum pulsní vlny [26, 33], dále 25 % maxima, bod největšího sklonu náběžné hrany aj. Bod v PPG signálu je vždy volen dle použitého algoritmu.

3.2 Senzory chytrého telefonu

Z nepřeberného množství senzorů, kterými jsou chytré telefony vybaveny, je pro měření krevního tlaku vhodné využít kombinaci mikrofonu a fotoaparátu, pomocí nichž jsme schopni získat PPG a FKG signál. Tyto konkrétní senzory byly zvoleny na základě prací zabývající se touto problematikou.[26, 33, 40]

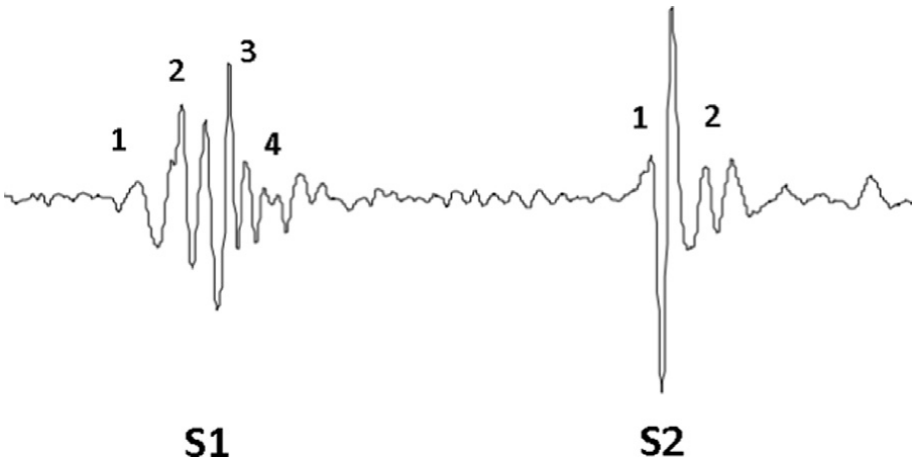
Výpočet PTT zahrnuje získání času příchodu pulsní vlny (Pulse Arrival Time - PAT) u obou bodů v tomtéž arteriálním řečišti, mezi kterými je PTT měřen. Jakmile jsou tyto časy zaznamenány je možné vypočítat PTT dle vztahu 3.2.[26]

$$PTT = PAT_1 - PAT_0 \tag{3.2}$$

3.2.1 Fonokardiografie

Proximální[1] bod, který odpovídá PAT_0, je v tomto případě získán ze signálu FKG pomocí mikrofonu přiloženého na hrudník uživatele. Lokalizace srdečních ozev je náročná, ale pro výpočet hodnoty krevního tlaku naprosto nezbytná. Během srdečního cyklu vznikají vibrace způsobené mechanickou aktivitou (otevírání a zavírání srdečních chlopní) šířící se skrz hrudník. Srdce produkuje převážně čtyři srdeční ozvy. U zdravého člověka je velmi dobře rozpoznatelná první (S1) a druhá (S2) srdeční ozva. Jednotlivé komponenty S1 a S2 jsou vyobrazeny na obrázku 3.2. Lokalizace srdečních chlopní a je poté vyobrazena na obrázku 3.3.[26, 29, 33, 40]

První srdeční ozva se skládá ze čtyř hlavních komponent. První komponenta S1 (1) odpovídá kontrakci komor, jež zvyšuje tlak, a tak tlačí krev směrem k síním, což způsobí uzavření cípatých chlopní (mitrální a trikuspidální). Uzavření cípatých chlopní odpovídá druhé komponentě S1 (2). Třetí komponenta S1 (3) odpovídá oscilacím krve při otevření poloměsíčitých chlopní (aortální a pulmonální). Čtvrtá komponenta S1 (4) odpovídá turbulencím způsobených krví proudící skrze aortu. Druhá srdeční ozva se skládá ze dvou složek (1 a 2), které odpovídají uzavření poloměsíčitých chlopní (aortální a pulmonální).[26]

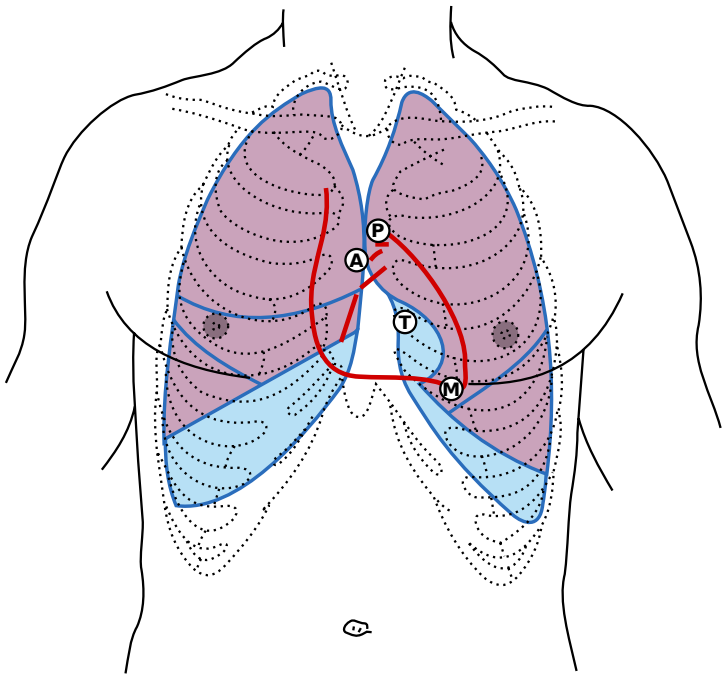


Obr. 3.2: Komponenty první a druhé srdeční ozvy.[26]

[1] proximální – bližší počátku či vzniku

Vzniku pulsní vlny odpovídá třetí a čtvrtá komponenta S1, při které je vypuzována krev do arteriálního řečiště. Tento okamžik odpovídá PAT_0.[26]

Jako nejlepší místo pro snímání FKG signálu se ukázala mitrální oblast. Tato oblast je charakteristická přijatelným držením telefonu při měření záznamu PPG, dostatečnou amplitudou S1 i S2, která je důležitá pro výpočet diastolického tlaku krve.[33]



Obr. 3.3: Lokalizace srdečních chlopní a jejich ozev. První srdeční ozva je produkována uzavřením mitrální (M) a trikuspidální (T) chlopně a otevřením aortální (A) a pulmonální (P) chlopně. Druhá srdeční ozva je produkována uzavřením aortální (A) a pulmonální (P) chlopně.[33]

3.2.2 Fotopletysmografie

Distální[2] bod, který odpovídá PAT_1, je v tomto případě získán ze signálu PPG za pomoci fotoaparátu a LED umístěné vedle objektivu. Oproti signálu FKG je získání PPG signálu poměrně snadné. Vyšetřovaná osoba přiloží koneček prstu současně přes objektiv i LED. Světlo emitované LED je rozptýleno, odraženo a absorbováno při průchodu živou tkání. Výrazné změny jasových hodnot jsou pak zachyceny snímačem fotoaparátu při různém množství krve v konečku prstu (průchod pulsní vlny). Na základě změn jasových hodnot jednotlivých snímků v čase je získán PPG signál.[33]

[2]distální – vzdálený, umístěný na opačné straně, než je počátek

3.3 Výpočet hodnoty krevního tlaku

Při stanovení hodnoty krevního tlaku vycházíme ze zkušeností získaných z prací [33], [26] a [40]. Tyto práce se zabývají stejnou problematikou.

V práci [33] je podrobně rozebrána metoda pro stanovení hodnoty krevního tlaku pomocí dvou telefonů. Autoři nejprve popisují způsob synchronizace mezi oběma zařízeními přes Bluetooth. První zařízení snímá FKG signál pomocí mikrofonu z hrudníku vyšetřované osoby. Druhé zařízení snímá PPG signál pomocí fotoaparátu a přisvětlovací LED z konečku prstu. Hodnota PTT je stanovena na základě S1 signálu FKG a vrcholu pulsu v PPG signálu. Na základě PTT je poté stanoven systolický tlak dle vztahu 3.3. Diastolický tlak je poté odhadnut na základě doby trvání vypuzovací fáze srdečního cyklu.

Práce [26] a [40] poté představují vylepšení v podobě využití pouze jednoho zařízení. Princip je téměř totožný s předchozí prací. Práce [26] velmi podrobně popisuje vývoj aplikace na operační systém *Android*. I přesto, že jsou oba signály (PPG a FKG) snímány stejným zařízením, jejich synchronizace je problematická. Autoři zde popisují princip synchronizace na základě poklepu konečkem prstu. Každé klepnutí přes objektiv fotoaparátu způsobí jeho zakrytí a zároveň generuje zvuk, jenž je zaznamenán. Současně tak vznikne vrchol v PPG i FKG signálu. Pokud jsou oba signály správně zarovnány vznik vrcholů je ve stejném časovém okamžiku.

Pokud není PPG signál **přesně** synchronizovaný s FKG signálem vznikne nepřesnost při výpočtu PTT, což způsobí nesprávný výpočet hodnoty krevního tlaku. Synchronizace PPG a FKG signálů je popsána v sekci 4.7. Pokud jsou signály vzájemně synchronizovány lze stanovit hodnotu krevního tlaku.[33]

3.3.1 Systolický arteriální tlak

V prvé řadě je nutné vypočítat PTT dle vztahu 3.2. Kde PAT_0 odpovídá S1 a PAT_1 odpovídá vrcholu pulsní vlny v PPG signálu. Poté lze systolický tlak vypočítat dle vztahu 3.3, který vznikl na základě lineární regrese z více než 500 měření dle [33].

Tento vztah odpovídá rovnici přímky ve směrnicovému tvaru $y = kx + q$, kde $-0,425$ je směrnice přímky k a konstanta $+214$ odpovídá q, jež určuje průsečík přímky s osou y. Konstanta $+214$ byla stanovena v [33] z výsledků naměřených hodnot systolického tlaku a PTT, které byly proloženy přímkou, aby byl získán regresní vztah, ze kterého lze získat tuto konstantu. Tato konstanta je vztažená k metodě popsané v [33]. Pro účely této práce bude nutné stanovit novou konstantu q, ale zachovat směrnici k.

$$p_S\ [mmHg] = -0,425 \cdot PTT + 214 \tag{3.3}$$

3.3.2 Diastolický arteriální tlak

Pro odhad diastolického tlaku je nutné znát základní údaje o uživateli, kterými jsou hmotnost, výška a věk. Nejprve je třeba stanovit systolický objem (Stroke Volume - SV). Ten lze vypočítat dle vztahu 3.4. Za klidových podmínek u zdravého dospělého člověka se SV pohybuje kolem hodnoty 70 ml.[29, 33]

$$SV\ [ml] = -6,6 + 0,25 \cdot (ET - 35) - 0,62 \cdot HR + 40,4 \cdot BSA - 0,51 \cdot věk \quad (3.4)$$

Kde ET je doba vypuzení (Ejection Time - ET), jež odpovídá době trvání vypuzovací fáze srdečního cyklu, který začíná S1 a končí S2. ET tedy lze vypočítat dle vztahu 3.5. Dalším parametrem potřebným k výpočtu SV je BSA. Povrch těla (Body Surface Area - BSA) je vypočítán dle vztahu 3.6, kde hmotnost je zadávána v kilogramech a výška v centimetrech. Posledním parametrem je HR, jenž odpovídá tepové frekvenci (Heart Rate - HR), kterou lze vypočítat dle vztahu 3.7, kde n je počet detekovaných pulsů a $\triangle t$ je délka záznamu v sekundách.

$$ET\ [ms] = S2 - S1 \quad (3.5)$$

$$BSA\ [m^2] = 0.007184 \cdot hmotnost^{0.425} \cdot výška^{0.725} \quad (3.6)$$

$$HR\ [tep/min] = \frac{n \cdot 60}{\triangle t} \quad (3.7)$$

Po výpočtu systolického objemu je možné vypočítat pulsní tlak dle vztahu 3.8. Tlak diastolický lze pak vypočítat vyjádřením ze vztahu 1.2 jako rozdíl tlaku systolického a pulsního.[33]

$$p_P\ [mmHg] = \frac{SV}{(0,013 \cdot hmotnost - 0,007 \cdot věk - 0,004 \cdot HR) + 1,307} \quad (3.8)$$

3.4 Volba chytrého telefonu

Aby bylo možné PPG a FKG signál zaznamenávat, je nutné, aby zvolené zařízení splňovalo minimální požadavky kladené těmito signály.

3.4.1 Minimální požadavky na zařízení

Krom splnění minimálních požadavků pro záznam PPG a FKG signálů, je vyžadováno, aby zařízení bylo dostatečně výkonné. Díky tomu je možné získaná data v reálném čase zpracovat a poskytnout tak uživateli zpětnou vazbu.

Při experimentech bylo zjištěno, že poskytnutí zpětné vazby značně napomáhá umístit potřebné senzory na správné místo, ze kterého lze signály snímat v požadované kvalitě. Tato zpětná vazba je nezbytná pro lokalizaci správného místa pro měření FKG signálu. Navíc audiovizuální zpětná vazba obohacuje uživatelskou zkušenost, což činí tuto metodu přitažlivější.[26]

Fotoaparát

Požadavky kladené na fotoaparát, kterým je snímán PPG signál, jsou splněny téměř každým chytrým telefonem. Ten musí být schopen zaznamenat **minimálně** 22 snímků za sekundu (frame per second - fps). Většina energie PPG signálu se nachází v nízkofrekvenčním složkách v rozsahu 5–11 Hz. Aby nebyl porušen vzorkovací teorém je tedy nutné zajistit minilální vzorkovací frekvenci 22 Hz, jinak dojde k neobnovitelné ztrátě dat.[26]

Minimální požadavky nesplňují zařízení, jenž mají přisvětlovací diodu umístěnou příliš daleko od objektivu fotoaparátu či nemají přisvětlovací diodu vůbec. Tím je znemožněno současné přiložení konečku prstu přes objektiv a diodu zároveň. Bez prosvícení konečku prstu není možné PPG signál získat.

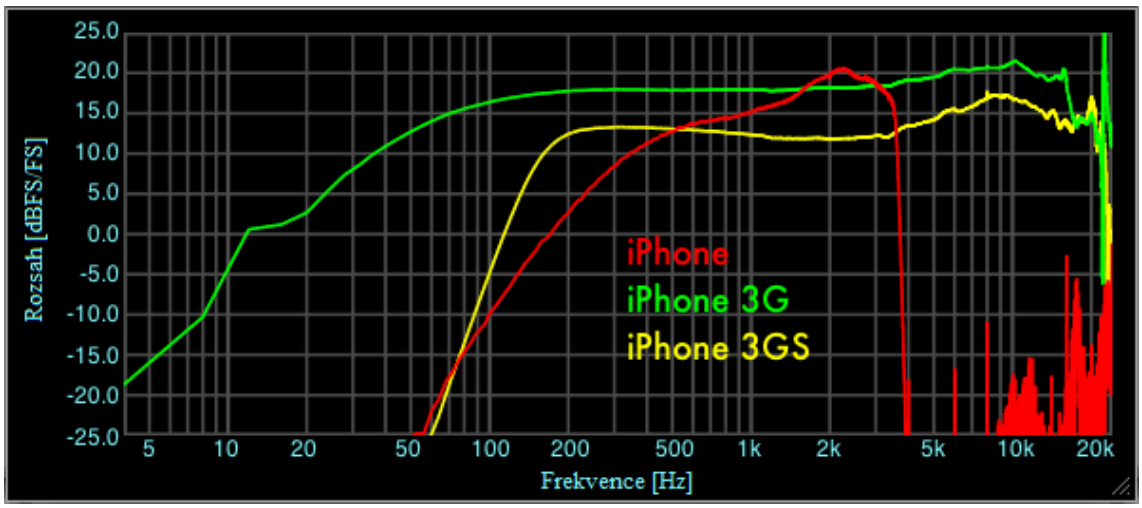
Rozlišení snímače ani další parametry nemají na kvalitu PPG záznamu značný vliv. Naopak je žádoucí snížit rozlišení video záznamu a tím snížit výpočetní čas algoritmu (méně dat ke zpracování).[33]

Mikrofon

Na rozdíl od fotoaparátu je mikrofon **základní vybavení** každého telefonu. Většina chytrých telefonů je vybavena více mikrofony - obvykle dvěma. Díky tomu je možné potlačit hluk okolí při hovoru či zaznamenávat stereo zvuk (například při nahrávní videozáznamu).[38]

Oblast slyšitelnosti se u člověka nachází v rozmezí 16 Hz až 20 kHz. Pro kvalitní zpracování akustických signálů je nutné zvolit mikrofon, jehož frekvenční charakteristika pokrývá oblast slyšitelnosti. Toho bohužel není vždy dosaženo. Pro výrobce

mobilních telefonů je velmi důležité, aby zvolený mikrofon pracoval velmi dobře v rozsahu od 250 Hz do 4 kHz. V tomto rozsahu se totiž nacházejí nejdůležitější složky řeči, na které je lidské ucho nejcitlivější. Na obr. 3.4 je vyobrazena frekvenční charakteristika mikrofonu různých zařízení od společnosti *Apple*. Z grafu je patrné, že zařízení *iPhone* a *iPhone 3GS* nejsou vhodné pro měření nízkých frekvenčních pásem.[38, 50]



Obr. 3.4: Frekvenční charakteristika mikrofonu různých zařízení.[38]

Frekvenční charakteristika mikrofonu, použitého v daném zařízení, není výrobcem **běžně udávaný parametr**. Proto je výběr zařízení **značně obtížný** - je nutné vyzkoušet, zdali zařízení vyhovuje.

Frekvenční spektrum srdečních ozev se nachází převážně v rozmezí 10–400 Hz. Z toho důvodu je nutné zajistit vzorkovací frekvenci minimálně 800 Hz, jinak dojde k porušení vzorkovacího teorému a tím k neobnovitelné ztrátě dat. Minimální vzorkovací frekvence je splněna **všemi** zařízeními (povahou telefonu je zpracovávat lidský hlas, jehož frekvence je několikanásobně vyšší). Problémem je ovšem citlivost mikrofonu na nízkofrekvenční složky. Ze zařízeních uvedených na obr. 3.4 těmto požadavkům vyhovuje pouze *iPhone 3G*, jenž je jako jediný schopný zaznamenávat i nízkofrekvenční složky audio signlálu. Pro záznam audio signálu jsou nejčastěji používány vzorkovací frekvence 44,1 kHz a 48 kHz. Z toho důvodu je vhodné, pro snížení výpočetního času, audiozáznam podvzorkovat na frekvenci blízkou 800 Hz.[26, 33, 38]

3.4.2 Zvolené zařízení

Výše uvedeným požadavkům **vyhovuje** například zařízení *Galaxy S7* od společnosti *Samsung*. Důvodem volby tohoto zařízení byla schopnost mikrofonu zaznamenávat

nízkofrekvenční složky audio signálu. Tato skutečnost byla zjištěna **empiricky** - parametry mikrofonu nebyly nalezeny na stránkách výrobce ani přidružených fórech.

Jedná se o výkonné a kvalitně vybavené zařízení. Výkon zařízení zajišťuje osmijádrový procesor *Samsung Exynos 8890* o frekvenci 2,3 GHz v kombinaci se 4 GB paměti RAM. Primární fotoaparát má rozlišení 12,2 Mpx a je tedy nadmíru dostačující. Umístění LED je v přijatelné vzdálenosti objektivu fotoaparátu. Zadní strana fotoaparátu je vyobrazena na obrázku 3.5.[51]

Obr. 3.5: Chytrý telefon *Samsung Galaxy S7*[51]

Jak je zmíněno výše, PPG signál lze poměrně snadno získat. Naopak získání FKG signálu je velice problematické. Záznam je velice citlivý na okolní rušení, a proto je nutné nahrávat v klidné místnosti. Dalším problémem jsou pohyby vyšetřované osoby. Pohyb mikrofonu po kůži má za následek neobnovitelné zničení záznamu. Tyto pohyby jsou obvykle způsobeny přiložením prstu při měření PPG signálu a změnou objemu hrudi vlivem dýchání. Aby se zabránilo vzniku nekvalitního záznamu je během měření nutné poskytnout vyšetřované osobě zpětnou vazbu v podobě audio-vizuální reprezentace. Signál FKG je možné zesílit a v reálném čase přehrávat, což ovšem vyžaduje příslušenství v podobě sluchátek. Další možností je zobrazit PPG a FKG signály na displeji telefonu. Zpětnou vazbu ovšem nelze **zajistit, bez aplikace vytvořené pro tento konkrétní účel**.

4 Vývoj mobilní aplikace

Na základě problematického snímání obou signálů, popsaných v předchozí kapitole, je žádoucí vytvořit aplikaci usnadňující akvizici dat. Toho je dosaženo zajištěním zpětné vazby v reálném čase uživateli, který signály měří. Podoba zpětné vazby PPG signálu je vizuální. Jedná se o vyobrazení grafu průměrných jasových hodnot červené složky jednotlivých snímků. Zpětnou vazbu FKG signálu lze realizovat vizuálně v podobě grafu či v podobě audia přehrávaného skrze připojená sluchátka. V ideálním případě kombinací obou - audiovizuálně.

4.1 Založení projektu

Pro vývoj aplikace bylo zvoleno vývojové prostředí *Android Studio*. Založení nového projektu je velmi jednoduché a intuitivní. Při jeho zakládání je nutné zvolit nejnižší verzi operačního systému, na kterou lze aplikaci nainstalovat a poté spustit. Na základě využití jednotlivých verzí, uvedených v tabulce 2.1, byla zvolena verze 6.0, tedy *Android Marshmallow*. Při vývoji se ovšem ukázalo, že tato verze není pro účely aplikace dostačující. Z tohoto důvodu bylo nutné celý projekt povýšit na verzi 7.0, tedy *Android Nougat*. Hlavním důvodem byla nemožnost zajištění časové známky u FKG signálu, která je nutná pro synchronizaci s PPG signálem. Po zvýšení minimální verze bylo nutné provést revizi kódu a opravit části, jež se staly zastaralé či nefunkční.

4.2 Architektura aplikace

Programová část projektu se sestává z jedné aktivity, dvou tříd implementovaných v programovacím jazyce *Java* a nativního skriptu na platformě *RenderScript*. Pro pochopení architektury projektu je nutná alespoň základní znalost objektově orientovaného programování (Object-Oriented Programming - OOP) a práce ve vláknech.

Třídy a aktivity

První ze dvou tříd nese název *VideoDataAcquisition*. Primární účel této třídy je práce s fotoaparátem telefonu, nahrání video záznamu, zpracování snímků v reálném čase a poskytnutí zpětné vazby.

Druhá třída nese název *AudioDataAcquisition*. Její primární účel je nahrávání audio záznamu skrz mikrofon telefonu, poskytnutí zpětné vazby a synchronizace audio záznamu s video záznamem.

Aby bylo možné využít funkcionalitu výše uvedených tříd je nutné nejprve vytvořit jejich instanci. Instance třídy je v OOP konkrétní datový objekt uložený v paměti vytvořený na základě vzoru - třídy. Ten se nachází v počátečním stavu definovaný konstruktorem - zavolaná metoda při instanci třídy. Inicializace těchto tříd je provedena v hlavní spouštěcí aktivitě.

Tato aktivita byla vytvořena při založení projektu, a proto nese název *MainActivity*. Jedná se o startovní bod aplikace, kde při jejím spuštění naběhne právě tato aktivita. Ta zkontroluje oprávnění a připraví GUI. V případě, že aplikace nemá dostatečná oprávnění je uživateli nabídnuto jejich udělení. Oprávnění, jež aplikace vyžaduje, jsou pro její běh aplikace esenciální. Jmenovitě se jedná oprávnění používat fotoaparát, mikrofon a zapisovat do souborového systému. Bez udělení těchto oprávnění je aplikace **bezpodmínečně ukončena**.

Výhodou výše uvedeného členění kódu je nezávislost tříd *VideoDataAcquisition* a *AudioDataAcquisition* na ostatních částech aplikace. Na základě těchto tříd lze tedy celkem pohodlně vytvořit aplikaci, která zpracovává stejná data, ale má jiný výstup. Například na základě třídy *VideoDataAcquisition* lze vytvořit aplikaci měřící tepovou frekvenci.

Optimalizace výkonu

Z důvodu výkonnostní náročnosti některých operací (nahrávání audia a videa) bylo nutné přesunout zátěž z hlavního vlákna (*Main Thread*), které je zodpovědné za veškeré události dějící se na obrazovce telefonu, do samostatných vláken. Spouštění náročných operací v tomto vlákně způsobí, že uživatel může pozorovat zadrhávání, zpoždění a nedostatečně citlivou odezvu na vstup (např. kliknutí na tlačítko). Pokud je hlavní vlákno zablokováno na více než pět sekund systém umožní uživateli aplikaci ukončit přímo. Na tomto základě bylo pro každou třídu vytvořeno samostatné vlákno běžící na pozadí - *VideoThread* a *AudioThread*.[14]

4.3 Kompatibilita

Při vývoji aplikace je velmi důležité seznámit se s doporučenými postupy, podporovanými formáty a technickými detaily. Hlavním důvodem je kompatibilita mezi různými druhy zařízení. Ty se vzájemně liší použitým HW při jejich výrobě. Proto s každou novou verzí operačního systému *Android* vychází dokument definice kompatibility (Compatibility Definition Document - CDD). Tento dokument uvádí požadavky, které **musí být splněny**, aby byla zařízení kompatibilní s nejnovější verzí systému Android. Dokument je důležitý jak pro výrobce, tak pro vývojáře (programátora). Výrobce musí zajistit, aby zařízení bylo HW kompatibilní s verzí operačního

systému na kterou cílí. Vývojář má díky dokumentu dostupné informace, jaká funkcionalita **musí být splněna** verzí operačního systému, na kterou aplikaci vyvíjí. Na základě CDD, verze 7.0, byla stanovena následující kritéria pro snímání PPG a FKG signálu. Význam neznámých tříd a zkratek je vysvětlen v odpovídajícím kontextu této kapitoly.[2]

Požadavky video signálu

- Všechny video kodéry a dekodéry **musí podporovat** barevný formát *YUV420*.
- Podpora video kodeku *H.264* (pro kódování i dekódování) je **vyžadována** všemi zařízeními.
- *Camera2 API* **musí podporovat** formát *YUV_420_888* a *JPEG* jako výstup pro třídu *ImageReader*.[2]

Požadavky audio signálu

- Podpora audio formátu *PCM* a *WAVE* **je vyžadována** všemi zařízeními.
- Pro záznam syrového audio signálu **musí** zařízení podporovat tyto vlastnosti:
  - **Formát**: *PCM*
  - **Bitová hloubka**: 16 bit
  - **Vzorkovací frekvence**: 8 000, 11 025, 16 000, 44 100 Hz
  - **Kanály**: Mono
- Záznam pro výše uvedené vzorkovací frekvence musí být proveden bez nadvzorkování či podvzorkování (HW řešení) a musí **zahrnovat** odpovídající **antialiasingový filtr**.[2]

4.4 Oprávnění

Aby bylo možné využívat funkčnost níže uvedených tříd, je nutné, aby byla aplikaci přidělena dostatečná oprávnění. Oprávnění aplikace se definuje v souboru *AndroidManifest.xml*. Potřebná oprávnění jsou uvedena ve výpisu 4.1.[3, 12, 20]

```
1 <!-- Používání fotoaparátu -->
2 <uses-permission android:name="android.permission.CAMERA" />
3 <!-- Nahrávání audio záznamu -->
4 <uses-permission android:name="android.permission.RECORD_AUDIO" />
5 <!-- Nahrávání video záznamu -->
6 <uses-permission android:name="android.permission.RECORD_VIDEO" />
7 <!-- Zápis do souborového systému -->
8 <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
```

Výpis 4.1: Oprávnění potřebná pro zápis a akvizici dat

4.5 Akvizice fotopletysmografických dat

Jak již bylo naznačeno, třída *VideoDataAcquisition* sdružuje metody a objekty, skrze které lze ovládat fotoaparát telefonu, zpracovávat jednotlivé snímky a poskytovat informace o aktuálním stavu snímané osoby. Tato sekce popisuje funkci této třídy.

4.5.1 Ovládání fotoaparátu

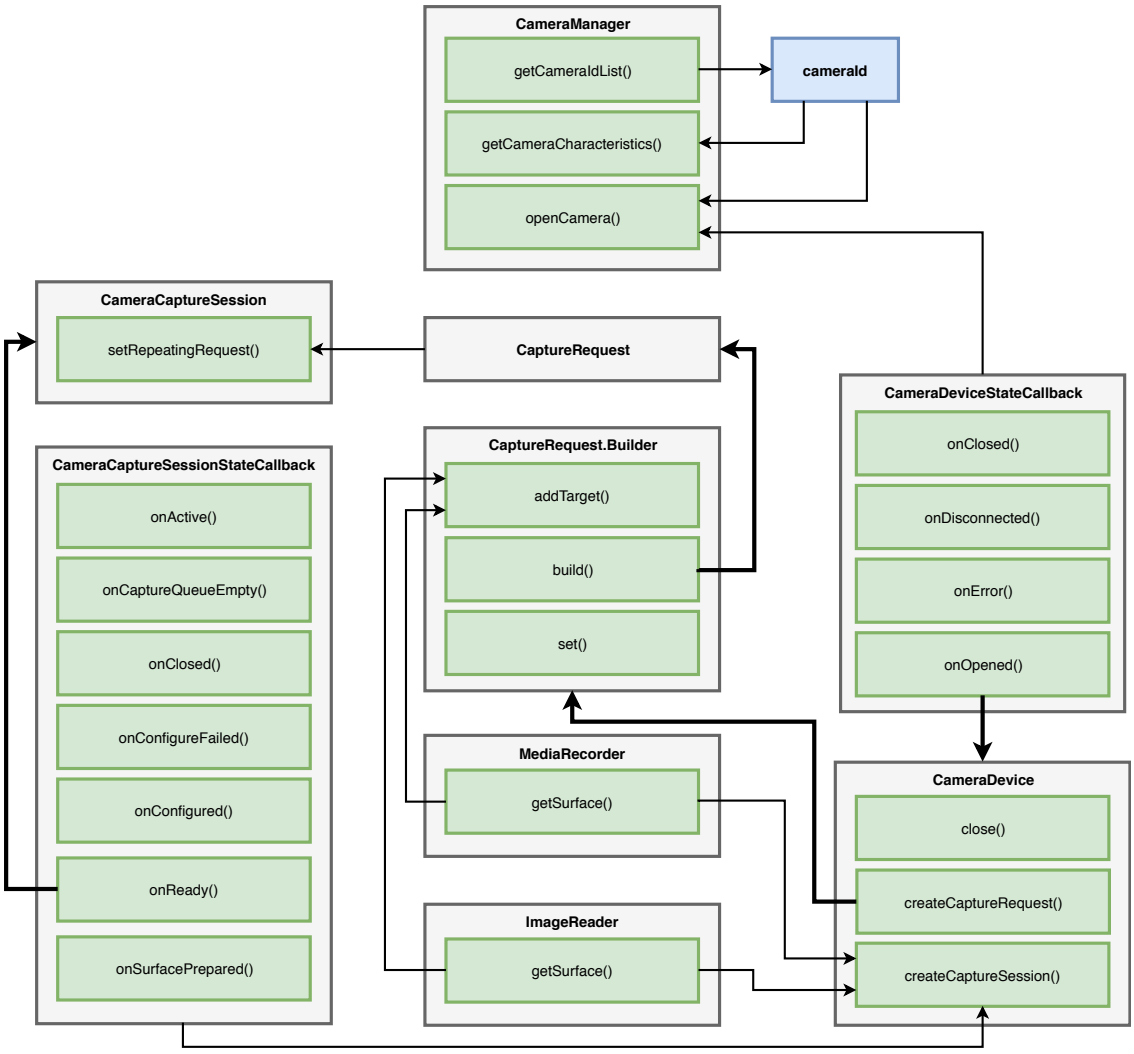
Vývojový diagram ovládání fotoaparátu je vyobrazen na obr. 4.1. Jeho dílčí části jsou podrobně rozebrány v následujících podsekcích. Průběh diagramu popisuje stav od otevření fotoaparátu až po získání po obrazových dat. Třídy *MediaRecorder* a *ImageReader* jsou popsány v sekci 4.5.2.

CameraManager

Klíčovým prvkem *Camera2 API* je třída *CameraManager*. Ta zajišťuje detekci, charakterizaci a zprostředkování přístupu k jednotlivým fotoaparátům. Použitím metody *getCameraIdList()* lze získat identifikátor všech aktuálně dostupných fotoaparátů. Na základě identifikátoru lze pak získat dodatečné informace o konkrétním fotoaparátu pomocí metody *getCameraCharacteristics()*. Jedná se pouze o výpis funkcí, kterými fotoaparát disponuje - jsou tedy neměnitelné. Pro účely této aplikace lze tedy zjistit, zdali vybraný fotoaparát disponuje přisvětlovací diodou či s jakými rozlišeními je schopen pracovat. Poslední, zde zmíněnou, metodou je metoda *openCamera()*, jež vytvoří přístup ke konkrétnímu HW fotoaparátu. Vstupem této metody je identifikátor fotoaparátu a instance třídy *CameraDeviceStateCallback*. Ukázka práce se třídou *CameraManager* je uvedena ve výpisu 4.2.[3, 19, 20, 25]

```
1  // Instance třídy CameraManager pomocí systémové služby
2  mCameraManager = (CameraManager)
       appContext.getSystemService(appContext.CAMERA_SERVICE);
3
4  // Identifikátor hlavního fotoaparátu
5  mCameraId = mCameraManager.getCameraIdList()[0];
6
7  // Informace o fotoaparátu
8  mCharacteristics = mCameraManager.getCameraCharacteristics(mCameraId);
9
10 // Zjištění přítomnosti přisvětlovací diody
11 boolean flash = mCharacteristics.get(CameraCharacteristics.FLASH_INFO_AVAILABLE)
12
13 // Otevření fotoaparátu na konkrétním vlákně (mBackgroundHandler)
14 mCameraManager.openCamera(mCameraId, mCameraDeviceStateCallback,
       mBackgroundHandler);
```

Výpis 4.2: Ukázka práce se třídou *CameraManager*



Obr. 4.1: Vývojový diagram ovládání fotoaparátu. Modré bloky reprezentují proměnné. Šedé bloky reprezentují instanci konkrétní třídy. Uvnitř těchto bloků se nacházejí důležité veřejné metody (zelené bloky) této třídy. Tučné šipky označují inicializaci dané třídy z konkrétní metody třídy jiné. Klasické šipky označují vstupy a výstupy metod.

CameraDeviceStateCallback

Instance třídy *CameraDeviceStateCallback* je návratový objekt (callback object) reagující na změnu stavu fotoaparátu. Změna stavu fotoaparátu vyvolá zavolání konkrétní metody obsluhující tento stav. Metody jsou následující.[18]

- *onClosed()* - zavoláno po zavření fotoaparátu
- *onDisconnected()* - zavoláno pokud již není fotoaparát k dispozici
- *onError()* - zavoláno při vážné chybě (např. chybě při otevírání fotoaparátu)
- *onOpened()* - zavoláno po otevření (skrz *CameraManager*)

Jelikož se jedná o práci s HW, je tedy **nutné využít** výše uvedené stavy. Důvodem je fakt, že po zavolání metody *openCamera()* z instance třídy *CameraManager*, vyžaduje fotoaparát určitý čas, než-li se otevře. Po jeho úspěšném otevření je zavolána metoda *onOpened()*, jejíž vstupní parametr je instance třídy *CameraDevice*. Ukázaka metody *onOpened()* je uvedena ve výpisu 4.3.[3, 18]

CameraDevice

Instance třídy *CameraDevice* reprezentuje již konkrétní fotoaparát. Před započetím snímání je ovšem nutné alokovat vyrovnávací paměť (buffer). Ten je v tomto případě reprezentován třídou *Surface*. Tato vyrovnávací paměť je obvykle vytvořena svými spotřebiteli - v případě této aplikace třídou *MediaRecorder* a třídou *ImageReader*, za pomoci metody *getSurface()*. Jinými slovy *MediaRecorder* a *ImageReader* vytvoří vyrovnávací paměť vyhovující jejich požadavkům. Dále je nutné vytvořit relaci snímání, které jsou přiřazeny jednotlivé vyrovnávací paměti. Ty jsou poté plněny obrazovými daty z konkrétního fotoaparátu - *CameraDevice*. Novou relaci snímání lze vytvořit pomocí metody *createCaptureSession()*, jejímiž vstupními parametry jsou jednotlivé vyrovnávací paměti a instance třídy *CameraCaptureSessionStateCallback*. Ukázka vytvoření nové relace je uvedena ve výpisu 4.3.[3, 17, 52]

```
1 public void onOpened(CameraDevice cameraDevice) {
2 // Přidání instancí vyrovnávacích paměti do listu
3    List<Surface> surfaces = new ArrayList<>();
4    surfaces.add(mMediaRecorder.getSurface());
5    surfaces.add(mImageReader.getSurface());
6
7 // Relace snímání na konkrétním vlákně (mBackgroundHandler)
8    cameraDevice.createCaptureSession(surfaces, mCameraCaptureSessionStateCallback,
         mBackgroundHandler);
9 }
```

Výpis 4.3: Ukázka metody *onOpened()* třídy *CameraDeviceStateCallback*

CameraCaptureSessionStateCallback

Podobně jako třída *CameraDeviceStateCallback* i tato třída implementuje návratový objekt reagující na změny stavu relace snímání. Její metody jsou následující.[16]

- *onActive()* - zavoláno při spuštění relace
- *onCaptureQueueEmpty()* - zavoláno při žádosti o další snímek
- *onClosed()* - zavoláno po ukončení relace
- *onConfigureFailed()* - zavoláno při nevyhovující konfiguraci *CameraDevice*
- *onConfigured()* - zavoláno po nakonfigurování *CameraDevice*
- *onReady()* - zavoláno vždy, když relace nemá žádnou další žádost o snímek
- *onSurfacePrepared()* - zavoláno při předběžném přidělení vyrovnávací paměti

Pokud nebudeme uvažovat chybové stavy, je nejprve zavolána metoda *onConfigured()*, jejíž zavolání je důkazem, že konfigurace *CameraDevice* byla úspěšná. Dále je zavolána metoda *onReady()*, ve které je nutné vytvořit žádost o zachycení snímku - *CaptureRequest*. *CaptureRequest* lze vytvořit pomocí metody *build()* třídy *CaptureRequest.Builder*. Ukázka metody *onReady()* je uvedena ve výpisu 4.4. [16, 21, 22]

Instanci třídy *CaptureRequest.Builder* lze získat pomocí metody *createCaptureRequest()* třídy *CameraDevice*. Jedná se o třídu zodpovědnou za stavbu a konfiguraci žádosti. Skrze metodou *addTarget()* jsou přiřazeny vyrovnávací paměti do níž budou odesílána obrazová data (je možné použít pouze vyrovnávací paměti, jež byly předány v metodě *createCaptureSession()*). Dále je zde možné nastavit stav přisvětlovací diody - pro využití v této aplikaci je dioda ve stavu svítilny (torch). Pokud je vše nakonfigurováno, pak zavoláním metody *build* je vytvořena instance třídy *CaptureRequest*.[21, 22]

Pro započetí snímaní je nutné zajistit nekonečné opakování (v případě video záznamu) získávání obrazových dat touto relací. Toho lze docílit zavoláním metody *setRepeatingRequest()* aktuální relace. Vstupem této funkce je již námi nakonfigurovaný *CaptureRequest*. Poté je zavolána metoda *onActive*, která značí začátek relace snímání. Uvnitř této metody je žádoucí zaznamenat časovou známku kvůli následné synchronizaci audio a video záznamu, tak aby bylo měření platné. Synchronizace je podrobně rozebrána v sekci 4.7.[16]

```
1  public void onReady(CameraCaptureSession session) {
2  // Instance třídy CaptureRequest.Builder dle vzoru CameraDevice.TEMPLATE_RECORD
3     CaptureRequest.Builder builder =
          mCameraDevice.createCaptureRequest(CameraDevice.TEMPLATE_RECORD);
4
5  // Rozsvícení přisvětlovací diody v režimu svítilny
6     builder.set(CaptureRequest.FLASH_MODE, CameraMetadata.FLASH_MODE_TORCH);
7
8  // Přiřazení vyrovnávacích pamětí
9     builder.addTarget(mImageReader.getSurface());
10    builder.addTarget(mMediaRecorder.getSurface());
11
12 // Sestavení žádosti
13    CaptureRequest request = builder.build();
14
15 // Vytvoření nekonečné smyčky žádostí na konkrétním vlákně (mBackgroundHandler)
16    session.setRepeatingRequest(request, null, mBackgroundHandler);
17 }
```

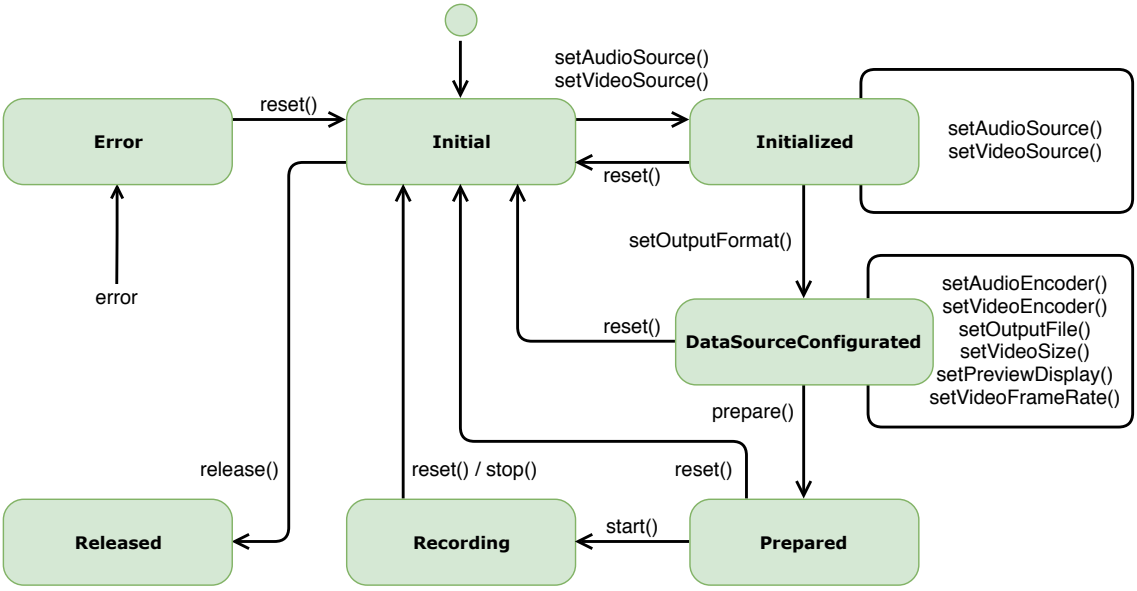
Výpis 4.4: Ukázka metody *onReady()* třídy *CameraCaptureSessionStateCallback*

4.5.2 Zpracování obrazových dat

Jakmile jsou vyrovnávací paměti plněny obrazovými daty, je vhodné data zpracovat. K tomuto účelu nám poslouží třídy *MediaRecorder* a *ImageReader*. V případě nahrávání běžného video záznamu je třída *MediaRecorder* naprosto dostačující. Ovšem v případě této aplikace je žádoucí uživateli zobrazit zpracovaná obrazová data v podobě PPG signálu. Třída *MediaRecorder* bohužel **neposkytuje** žádné informace o aktuálně zpracovaných snímcích. Proto bylo nutné využít třídu *ImageReader*, která **umožňuje** zpracovávat obrazová data přímo.[37, 44]

MediaRecorder

Instance této třídy je běžně využívána pro nahrávání audio a video záznamů. Vývojáři nabízí velké množství nastavení, které ovlivňuje kvalitu výsledného záznamu. Třída *MediaRecorder* pracuje na principu stavového automatu, jehož stavy a přechody mezi nimi jsou vyobrazeny na obrázku 4.2.[44]



Obr. 4.2: Vývojový diagram stavů a přechodů třídy *MediaRecorder* [44]

Před započetím nahrávání video záznamu je vyžadována konfigurace instance třídy *MediaRecorder*. Ta je uvedena ve výpisu 4.5. Po konfiguraci je zavolána metoda *prepare()*, jež připraví *MediaRecorder* na sběr a kódování dat. Po úspěšném průběhu lze využít metodu *getSurface()* pro získání vyrovnávací paměti. Pokud je fotoaparát otevřený, jsou přiřazeny vyrovnávací paměti a je vytvořena žádost snímání, pak je možné pomocí metody *start()* zahájit zpracování obrazových dat. Nahrávání je ukončeno zavoláním metody *stop()*. Po této metodě již nelze zavolat metodu *start()*

bez opětovné konfigurace. Dále metoda *reset()* přesune *MediaRecorder* do stavu nečinnosti. V poslední řadě metoda *release()* uvolní veškeré asociované zdroje touto třídou. Výstupem je video záznam ve formátu *MP4*.[44]

```
1  // Inicializace
2  mMediaRecorder = new MediaRecorder();
3
4  // Nastavení zdroje obrazových dat
5  mMediaRecorder.setVideoSource(MediaRecorder.VideoSource.SURFACE);
6
7  // Nastavení výstupního formátu souboru
8  mMediaRecorder.setOutputFormat(MediaRecorder.OutputFormat.MPEG_4);
9
10 // Nastavení cesty v souborovém systému, kde bude záznam uložen
11 mMediaRecorder.setOutputFile(mFilePath);
12
13 // Nalezení vyhovujícího rozlišení pro potřeby této aplikace
14 StreamConfigurationMap map =
       mCharacteristics.get(CameraCharacteristics.SCALER_STREAM_CONFIGURATION_MAP);
15 mVideoSize = chooseVideoSize(map.getOutputSizes(MediaRecorder.class));
16
17 // Nastavení vzorkovací frekvence - počtu snímku za sekundu
18 mMediaRecorder.setVideoFrameRate(VIDEO_FRAME_RATE);
19
20 // Nastavení rozlišení videa
21 mMediaRecorder.setVideoSize(mVideoSize.getWidth(), mVideoSize.getHeight());
22
23 // Nastavení kodéru
24 mMediaRecorder.setVideoEncoder(MediaRecorder.VideoEncoder.H264);
25
26 // Nastavení orientace video záznamu (90 odpovídá pozici na výšku)
27 mMediaRecorder.setOrientationHint(90);
```

Výpis 4.5: Konfigurace třídy *MediaRecorder*

ImageReader

Jelikož třída *MediaRecorder* neumožňuje přístup k jednotlivým snímkům, není tak možné získat PPG signál. Z toho důvodu je pro získání PPG signálu použita třída *ImageReader*. Ta umožňuje přímý přístup k obrazovým datům. Třída *ImageReader* je nakonfigurována při její inicializaci. Metoda *newInstance()* očekává ve vstupních parametrech šířku a výšku snímku, formát a maximální počet snímků zpracovávaných zároveň. Konfigurace třídy *ImageReader* je uvedena ve výpisu 4.6.

```
1  // Inicializace
2  mImageReader = ImageReader.newInstance(mVideoSize.getWidth(),
       mVideoSize.getHeight(), ImageFormat.YUV_420_888, 1);
3
4  // Nastavení zpracování nových snímků na konkrétním vlákně (mBackgroundHandler)
5  mImageReader.setOnImageAvailableListener(mOnImageAvailableListener,
       mBackgroundHandler);
```

Výpis 4.6: Konfigurace třídy *ImageReader*

Rozměry snímku jsou v tomto případě stejné jako v třídě *MediaRecorder*. Třída *ImageFormat* sdružuje veškeré dostupné formáty záznamu obrazových dat na platformě Android. Je důležité poznamenat, že některé formáty vyžadují konkrétní HW, a proto je nelze využít na některých zařízeních - viz CDD v sekci 4.3.

Dále je nutné pomocí metody *setOnImageAvailableListener()* přiřadit instanci třídy *ImageReader* návratové rozhraní. Toto rozhraní reaguje na nové snímky. Obsahuje pouze metodu *onImageAvailable()*, ve které je možné zpracovat obrazová data. Příklady této metody jsou uvedeny ve výpise 4.7 pro formát *JPEG* a ve výpise 4.9 pro formát *YUV_420_888*.

Formát JPEG

Prvním využitým formátem byl komprimovaný *JPEG*. Snahou při využití tohoto formátu bylo vyhnutí se zpracování obrazových dat na nativní úrovni. Abychom byli schopni stanovit průměrnou hodnotu červeného kanálu, bylo nutné převést *JPEG* na *Bitmap* formát. Tento převod je uveden ve výpisu 4.7. Třída *Bitmap*, jež reprezentuje tento formát, zahrnuje metodu *getPixel()*. Ta očekává na vstupu souřadnice pixelu. Díky tomu lze získat hodnoty všech pixelů pomocí dvou cyklů dle rozměru snímku. Výstupem metody je datový typ *integer*, jenž definuje barvu v barevném prostoru *sRGB*. Každá barva je složena ze tří barevných složek a alfa kanálu.[15, 36, 32, 23]

- **A**lpha kanál - průhlednost
- **R**ed - červená
- **G**reen - zelená
- **B**lue - modrá

Každá z výše uvedených složek nabývá celočíselné hodnoty mezi 0 a 255. To odpovídá 8 bitům na každou složku. Jelikož se pixel skládá z těchto 4 složek, je zapsán pomocí 32-bitového čísla. Hodnota průhlednosti je zapsána bity na pozici 31–24, červená na pozici 23–16, zelená na pozici 15–8 a modrá na pozici 7–0. Toto je vyobrazeno na obrázku 4.3. Získání průměrné jasové hodnoty červené složky jednotlivých pixelů konkrétního snímku je uvedeno ve výpisu 4.8.[32]

ALPHA								RED								GREEN								BLUE							
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0	7	6	5	4	3	2	1	0

Obr. 4.3: Pozice bitů jednotlivých složek barevného modelu ARGB [32]

Tento postup se ukázal značně nevyhovující. Hlavním problémem byla velmi nízká vzorkovací frekvence video záznamu. Při rozlišení 320×240 pixelů dosahovala přibližně 6 Hz. Takto rapidní snížení vzorkovací frekvence bylo zapříčiněno interním převodem obrazových dat do formátu *JPEG* a výpočtem průměrných jasových hodnot červené složky na tomto vlákně. Samotný převod do formátu *JPEG* snížil vzorkovací frekvenci z původních 30 Hz na 12 Hz, výpočet průměru dále z 12 Hz na 6 Hz. V sekci 3.4.1, je uvedeno, že minimální vzorkovací frekvence PPG signálu je 22 Hz, jinak dojde k porušení vzorkovacího teorému a tím k neobnovitelné ztrátě dat. Z toho důvodu není možné tento postup zpracovávání obrazových dat použít.

```
1  // Zpracování obrazových dat - formát JPEG
2  public void onImageAvailable(ImageReader reader) {
3     Image image =  reader.acquireNextImage();
4     ByteBuffer buffer = image.getPlanes()[0].getBuffer();
5     byte[] bytes = new byte[buffer.capacity()];
6     buffer.get(bytes);
7     Bitmap bitmap = BitmapFactory.decodeByteArray(bytes, 0, bytes.length, null);
8
9     // Výpočet průměrné jasové hodnoty červené složky
10    int redAvg = getAverageValueOfRedChannel(bitmap);
11    image.close();
12 }
```

Výpis 4.7: Ukázka metody *onImageAvailable* zpracovávající formát *JPEG*

```
1  // Výpočet průměrné jasové hodnoty červené složky snímku
2  private int getAverageValueOfRedChannel(Bitmap bitmap) {
3     int width = bitmap.getWidth();
4     int height = bitmap.getHeight();
5
6     // Suma jasových hodnot červené složky všech pixelů
7     int red = 0;
8
9     // Procházení jednotlivých pixelů
10    for (int x = 0; x < width; x++) {
11       for (int y = 0; y < height; y++) {
12          int pixel = bitmap.getPixel(x, y);
13
14          // Jasová hodnota červené složky barevného modelu ARGB
15          red += (pixel >> 16) & 0xff;
16       }
17    }
18
19    // Výpočet průměrné jasové hodnoty červené složky ve snímku
20    return red / (width * height); // Suma dělená počtem pixelů
21 }
```

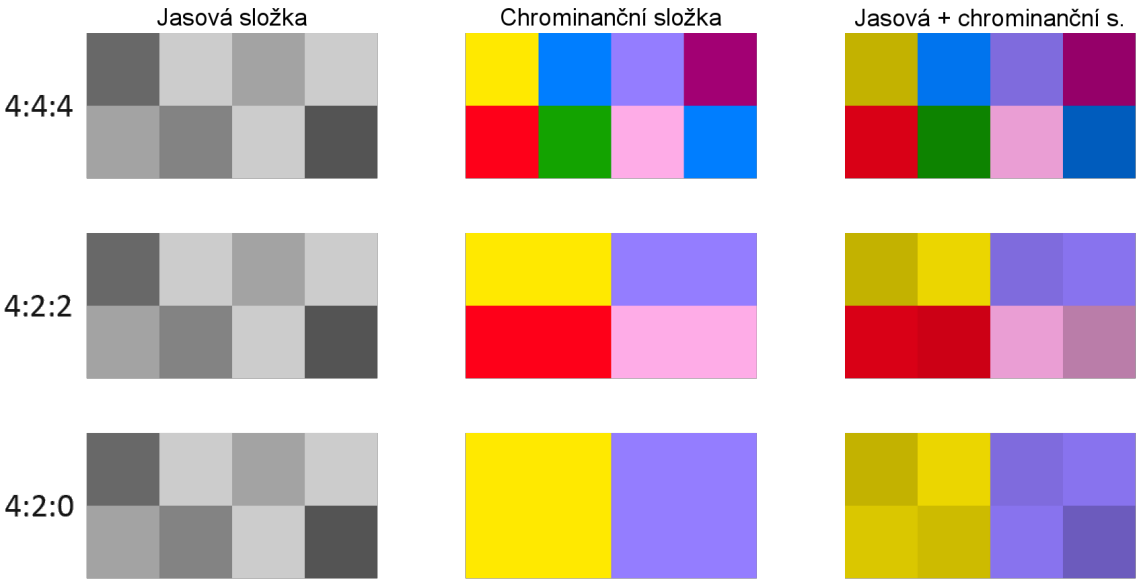
Výpis 4.8: Ukázka metody *getAverageValueOfRedChannel()*

Formát YUV

Po předchozí zkušenosti s formátem *JPEG* bylo zřejmé, že implementace zpracování obrazových dat v programovacím jazyce *Java* není vhodná, jelikož nesplňuje minimální požadavky pro snímání signálu PPG. Na tomto základě bylo nutné implementovat zpracování obrazových dat na nativní úrovni, protože zpracování formátu *YUV* je náročnější. Pro tyto účely je dokumentací [3] doporučen formát *YUV_420_888*.

Označení *YUV*, vztažené k digitálnímu zpracování, je obvykle používáno k popisu formátů kódovaných pomocí barevného modelu *YCbCr*. Formát *YUV_420_888* tedy reprezentuje obecný formát *YCbCr* pracující s oddělenou jasovou složkou *Y* a chrominančními komponenty definujícími barevný odstín - *Cb* reprezentuje modrou chrominanční složku a *Cr* reprezentuje červenou chrominanční složku.[35, 36]

Číslo *420* v názvu popisuje způsob podvzorkování barvonosných složek ve prospěch složky jasu, na kterou je lidské oko citlivější. Typičtější značení způsobu podvzorkování je ve formátu *J:a:b*, kde *J* určuje velikost vzorku, *a* určuje počet barvonosných složek prvního řádku a *b* určuje počet barvonosných složek druhého řádku. Jednotlivé příklady jsou vyobrazeny na obrázku 4.4. Schéma *4:4:4* zachovává vzorkovací frekvenci v obou směrech. Schéma *4:2:2* zachovává vertikální vzorkovací frekvenci, ale horizontální vzorkovací frekvence je snížena na polovinu. Schéma *4:2:0* snižuje vzorkovací frekvenci na polovinu v obou směrech.[35]



Obr. 4.4: Podvzorkování barvonosných složek formátu *YUV* [35]

Dále číslo *888* v názvu odpovídá 8 bitům na vzorek každé složky. Obrazová data v tomto formátu jsou vždy reprezentována třemi samostatnými barevnými plochami (plane). Pořadí těchto ploch v poli, vráceného metodou *getPlanes()* třídy *Image*, je garantováno - na pozici 0 nalezneme složku jasu Y, na pozici 1 složku U (Cb) a na pozici 2 složku V (Cr). Získání jednotlivých složek je uvedeno ve výpisu 4.9. Jednotlivé složky jsou uloženy v bytovém poli (vektorová reprezentace), proto je nutné získat dodatečné informace o způsobu jejich uložení pomocí metod *getRowStride()* a *getPixelStride()*. Jednotlivé metody vrací krok řádku či pixelu.[36]

```
1  // Zpracování obrazových dat - formát YUV_420_888
2  public void onImageAvailable(ImageReader reader) {
3     // Získání nejnovějších obrazových dat
4     Image image =  reader.acquireNextImage();
5
6     // Pole barevných ploch
7     Image.Plane[] planes = image.getPlanes();
8
9     // Deklarace bytové paměti
10    ByteBuffer buffer;
11
12    // Složka jasu
13    buffer = planes[0].getBuffer();
14    byte[] y = new byte[buffer.remaining()];
15    buffer.get(y);
16
17    // Modrá chrominanční složka
18    buffer = planes[1].getBuffer();
19    byte[] u = new byte[buffer.remaining()];
20    buffer.get(u)
21
22    // Červená chrominanční složka
23    buffer = planes[2].getBuffer();
24    byte[] v = new byte[buffer.remaining()];
25    buffer.get(v);
26
27    // Dodatečné informace o způsobu uložení složek výše - proměnné y, u, v
28    int yRowStride = planes[0].getRowStride();
29    int yPixelStride = planes[0].getPixelStride();
30    int uvRowStride = planes[1].getRowStride();      // u == v
31    int uvPixelStride = planes[1].getPixelStride();  // u == v
32
33    // RenderScript
34    // ............
35
36    // Uvolnění prostředků (došlo by k přetečení vyrovnávací paměti)
37    image.close();
38 }
```

Výpis 4.9: Ukázka metody *onImageAvailable* zpracovávající formát *YUV*

Z výše uvedeného vyplývá, že nejpřesnějšího výsledku bylo dosaženo při použití formátu *YUV_444_888*, jenž není podvzorkovaný. Bohužel zařízení, na kterém byla aplikace vyvíjena tento formát nepodporuje. Obdobně dopadl i formát

YUV_422_888. Na formát *YUV_420_888* je v dokumentaci nejvíce odkazováno. Tento formát musí být totiž podporovaný všemi zařízeními, jenž implementují *Camera2 API* - viz CDD 4.3.[36, 37]

Při použití formátu *YUV_420_888* a zpracování obrazových dat pomocí platformy *RenderScript* je dosaženo vzorkovací frekvence 30 Hz. Toto řešení tedy splňuje minimální požadavky pro akvizici PPG signálu.

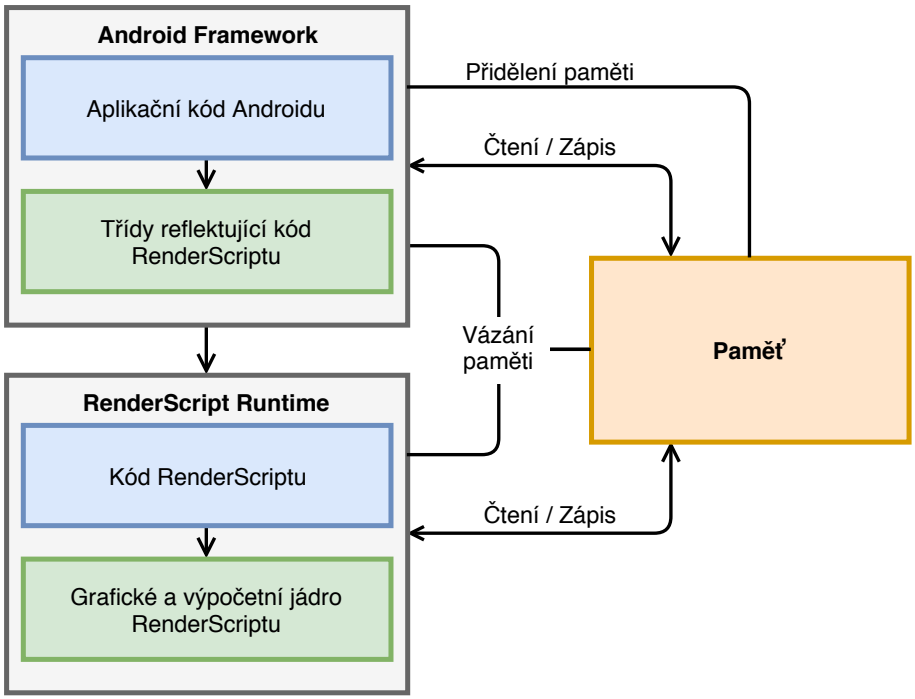
4.5.3 RenderScript

Jedná se o platformu založenou na *Androidu*, pracující na nativní úrovni. Využívá se pro velmi náročné výpočetní úlohy. Psaní kódu na platformě *RenderScript* se velmi podobá psaní kódu v programovacím jazyce *C* či *C++*. Je ovšem možné využít pouze funkce, které jsou definované v *RenderScript API*. Konkrétně se jedná o správu zdrojů a řízení jádra. API je dostupné ve třech různých jazycích - *Java*, *C++* a *C99* (verze programovacího jazyka *C*). Díky své rychlosti je často využíván v aplikacích zaměřujících se na některou z následujících operací.[48, 49]

- Zpracování obrazů (image processing)
- Výpočetní fotografie (computational photography)
- Počítačové vidění (computer vision)

Vysoké rychlosti je dosaženo díky paralelnímu výpočtu dat. Jinými slovy je pracovní zátěž rozložena na všechna jádra všech dostupných výpočetních jednotek - CPU, GPU či DSP. To umožňuje implementovat složitější algoritmy bez obav ze ztráty výkonu. Nízkoúrovňové běhové prostředí platformy *RednerScript* je ovládáno nadřazeným systémem - *Androidem*. Ten odpovídá za správu paměti - *Renderscript* pracuje pouze s pamětí, jež mu byla přidělena. Komunikace systému *Android* s platformou *RenderScript* je asynchronní - dotazy jsou umísťovány do fronty a jsou zpracovány jakmile je to možné. Práce a sdílení zdrojů systému *Android* s platformou *RenderScript* je vyobrazena na obrázku 4.5. Obvyklý postup pro využití platformy *RenderScript* z programovacího jazyka *Java* je následující.[48, 49]

1. Inicializovat kontext platformy *RenderScript*
2. Zarezervovat paměť a svázat jí (přidělit) platformou *RenderScript*
3. Vytvořit skript
4. Naplnit přidělenou paměť daty
5. Nastavit globální proměnné
6. Spustit příslušné funkce
7. Získat nová data z paměti
8. Zničit kontext platformy *RenderScript*



Obr. 4.5: Práce a sdílení zdrojů systému *Androidu* s platformou *RenderScript* [7]

Obrazová data jsou získávána ve formátu *YUV_420_888*. Barevný model *YUV* ovšem neobsahuje složky potřebné k získání PPG signálu přímo. Proto je nutné převést obrazová data z barevného modelu *YUV* na *ARGB* dle vztahu 4.1, kde jednotlivé složky Y, U, V odpovídají hodnotě konkrétního pixelu reprezentované 8-bitovým číslem (0–255). Složky A, R, G, B jsou taktéž reprezentovány 8-bitovým číslem (0–255), a proto je brána v potaz pouze celočíselná část výpočtu. Dále musí být rozsah výpočtu udržen v rozmezí 0–255. V případě, že je hodnota menší než 0, je rovna 0. V případě, že je větší než 255, je rovna 255. Pozice jednotlivých bitů barevného modelu *ARGB* jsou vyobrazeny na obrázku 4.3. Jelikož je PPG signál získán pouze z průměrné jasové hodnoty červené složky jednotlivých pixelů konkrétního snímku lze výpočet ostatních složek ignorovat. Tím je snížena výpočetní náročnost. V ideálním případě je výstupem algoritmu přímo průměrná jasová hodnota červené složky konkrétního snímku.[6, 49]

$$\begin{aligned} A &= 255 \\ R &= Y + V \cdot \frac{1436}{1024} - 179 \\ G &= Y - U \cdot \frac{46549}{131072} + 44 - V \cdot \frac{93604}{131072} + 91 \\ B &= Y + U \cdot \frac{1814}{1024} - 227 \end{aligned} \tag{4.1}$$

Výpočet jasové hodnoty červené složky konkrétního pixelu vyžaduje pro výpočet pouze složku jasu Y a červenou chrominační složku V (Cr). Z toho důvodu není nutné alokovat paměť pro složku U (Cb). Získání jasové hodnoty červené složky konkrétního pixelu a průměr těchto jasových hodnot konkrétního snímku je uveden ve výpise 4.10. Po získání jasové hodnoty červené složky konkrétního pixelu, pomocí metody *getRedFromYuv()*, je tato hodnota přičtena do globální proměnné *pixelSum*. Z této proměnné je poté vypočítán průměr jako podíl této proměnné počtem pixelů snímku ($šířka \cdot výška$) pomocí metody *getRedAvg()*. Před vstupem každého nového snímku je nutné vynulovat proměnnou *pixelSum* pomocí metody *reset()*.

```
1  #pragma version(1) // Verze RenderScriptu
2  #pragma rs java_package_name(cz.vutbr.bpdataacquisition) // Název balíčku
3  #pragma rs_fp_relaxed // Nastavení přesnosti desetinných míst
4
5  static int pixelSum = 0; // Suma jasových hodnot červené složky
6
7  uint width, height; // Rozměry snímku
8  uint vPixelStride, vRowStride; // Krok pixelů a řádků
9
10 rs_allocation ypsIn, vIn; // Přidělené paměti
11
12 // Získání jasových hodnot červené složky z YUV formátu
13 void __attribute__((kernel)) getRedFromYuv(uint32_t x, uint32_t y) {
14    // Výpočet pozice chrominanční složky na základě pozice složky jasu
15    uint uvIndex = vPixelStride * (x/2) + vRowStride * (y/2);
16
17    // Získání Y a V z paměti na konkrétních souřadnicích
18     uchar yps = rsGetElementAt_uchar(ypsIn, x, y);
19     uchar v = rsGetElementAt_uchar(vIn, uvIndex);
20
21    // Výpočet jasové hodnoty červené složky a udržení hodnoty v rozmezí 0-255
22     uchar red = clamp(yps + v * 1436 / 1024 - 179, 0, 255);
23
24     // Aktualizování hodnoty sdílené mezi vlákny - suma
25     rsAtomicAdd(&pixelSum, red);
26 }
27
28 // Výpočet průměrné jasové hodnoty červené složky snímku
29 int __attribute__((kernel)) getRedAvg(int x){
30     return pixelSum / (width * height);
31 }
32
33 // Vynulování hodnot, aby mohl být spočítán další snímek
34 void reset(){
35     pixelSum = 0;
36 }
```

Výpis 4.10: Získání PPG signálu z formátu *YUV* na platformě *RenderScript*

4.6 Akvizice fonokardiografických dat

Třída *AudioDataAcquisition* sdružuje metody a objekty, skrze které lze pracovat s mikrofonem zařízení, zpracovávat jednotlivé vzorky audio signálu a poskytovat zpětnou vazbu uživateli. Tato sekce popisuje funkci této třídy.

4.6.1 Ovládání zvukových zařízení

Akvizice audio signálu na operačním systému *Android* je oproti video signálu podstatně jednodušší. Pokud je nahrávání audio a video záznamu odděleno, je nutné provést vzájemnou synchronizaci. V této sekci jsou popsány jednotlivé třídy umožňující ovládání zvukových zařízení a záznam či přehrávání zvukových dat.

AudioRecord

Třída *AudioRecord* odpovídá za správu zvukových zdrojů. Při inicializaci této třídy jsou v konstruktoru očekávány parametry, jenž definují způsob záznamu zvukových dat. Parametry jsou následující.[12]

- Zdroj zvukových dat
- Vzorkovací frekvence
- Nastavení kanálů
- Kódování a bitovou hloubku
- Velikost vyrovnávací paměti

Instance třídy *AudioRecord* umožňuje přímý přístup ke zvukovým datům. Při jejím vytvoření si tato instance třídy inicializuje vyrovnávací paměť, která bude po spuštění nahrávání plněna zvukovými daty - vzorky. Velikost vyrovnávací paměti určuje, jak dlouho lze zaznamenávat zvuková data, než-li dojde k přetečení, a tak ke ztrátě dat.[12]

Nahrávání je zahájeno zavoláním metody *startRecording()*. Po zavolání této metody se vyrovnávací paměť začne plnit zvukovými daty. Ty je nutné neustále odebírat pomocí metody *read()*. Doporučeným způsobem je použití *while* cyklu, jehož podmínka je splněna, dokud není vnějším zásahem změněn její stav.[12]

Metoda *read()* umožňuje vracet zvuková data v různých datových typech (*byte[]*, *short[]*, *ByteBuffer*). Výběr správného datového typu závisí na dalším zpracování zvukových dat. Vzhledem k tomu, že pro převod analogového signálu na digitální používáme lineární pulzně kódovou modulaci (Pulse Code Modulation - PCM) o bitové hloubce 16 bitů, je vhodné využít datový typ *short*, jenž je také reprezentován 16 bity. Díky tomu **jeden vzorek odpovídá jedné hodnotě** v poli. V případě použití datového typu *byte* by byl jeden vzorek reprezentován dvěma hodnotami v

poli - dvěma byty. Toto řešení by ovšem zkomplikovalo úpravu amplitudy signálu. Práce se třídou *AudioRecord* je uvedena ve výpise 4.11.[10, 12]

Zpracování zvukových dat je ukončeno změnou podmínky *while* cyklu. Plnění vyrovnávací paměti je zastaveno pomocí metody *stop()*. Dále je vhodné uvolnit veškeré zdroje, jenž byly využívány, pomocí metody *release()*.[12]

```
1  int BUFFER_SIZE = 1024; // Velikost vyrovnávací paměti
2
3  // Inicializace
4  mAudioRecord = new AudioRecord(
5     MediaRecorder.AudioSource.MIC, // Zdroj audio signálu
6     8000, // Vzorkovací frekvence
7     AudioFormat.CHANNEL_IN_MONO, // Nastavení kanálů
8     AudioFormat.ENCODING_PCM_16BIT, // Nastavení kódování a bitové hloubky
9     BUFFER_SIZE * 2); // Velikost vyrovnávací paměti v bytech
10
11 // Začátek nahrávání
12 mAudioRecord.startRecording();
13 mIsRecording = true; // Stav nahrávání (pro while cyklus)
14
15 // Pole pro čtení dat z vyrovnávací paměti (pro metodu read())
16 short rawData[] = new short[BUFFER_SIZE];
17
18 // Zápis do souborového systému
19 FileOutputStream os = null;
20 os = new FileOutputStream(mFilePath + ".pcm");
21
22 // Zastavení cyklu vyžaduje změnu proměnné mIsRecording na hodnotu false
23 while (mIsRecording) { // Ukončeno například po kliknutí na tlačítko STOP
24    // Čtení zvukových dat z vyrovnávací paměti
25    mAudioRecord.read(rawData, 0, BUFFER_SIZE);
26
27    //Převod datového typu short na byte
28    byte byteData[] = short2byte(rawData);
29
30    // Zápis zvukových dat do souboru v bytech
31    os.write(byteData, 0, BUFFER_SIZE * 2);
32 }
33
34 mAudioRecord.stop(); // Ukončení nahrávání
35 mAudioRecord.release(); // Uvolnění zdrojů
```

Výpis 4.11: Ukázka práce se třídou *AudioRecord*

AudioManager

Primární účel této třídy je poskytnout přístup k ovládání hlasitosti médií a módům vyzvánění. Mimo jiné obsahuje velké množství konstant a informací o aktuálně připojených zvukových příslušenstvích. Toho je využito, aby se zabránilo připojení sluchátek se zabudovaným mikrofonem - tzv. headset. Zvuková data jsou poté získávána z obou mikrofonů zároveň (mikrofon zabudovaný v telefonu a mikrofon zabudovaný ve sluchátkách) v rámci jednoho kanálu, což není žádoucí. Mezi povolená zvuková

příslušenství, skrze které je uživateli poskytnuta zpětná vazba (přehrávání zvukových dat), patří drátová sluchátka a veškerá bluetooth zařízení podporující profil *A2DP* (bluetooth sluchátka, bluetooth reproduktor apod.) sloužící k bezdrátovému přenosu hudby. Zjištění, zdali je připojené některé z povoleného příslušenství, je uvedeno ve výpise 4.12. Na základě této metody je rozhodnuto, zdali bude využita třída *AudioTrack* pro přehrávání získaných zvukových dat.[9, 11]

```
1  private boolean isHeadphonesPlugged(){
2     // Získání všech zvukových zařízeních připojených k systému
3     AudioDeviceInfo[] audioDevices =
          mAudioManager.getDevices(AudioManager.GET_DEVICES_ALL);
4
5     // Zjištění typu zařízení
6     for(AudioDeviceInfo deviceInfo : audioDevices) {
7        if(deviceInfo.getType() == AudioDeviceInfo.TYPE_WIRED_HEADPHONES ||
             deviceInfo.getType() == AudioDeviceInfo.TYPE_BLUETOOTH_A2DP) {
8           // Pokud se jedná o drátová sluchátka či BT zařízení (A2DP)
9           return true;
10       }
11    }
12    return false; // Pokud není nalezeno odpovídající příslušenství
13 }
```

Výpis 4.12: Získání informací o připojených zvukových zařízeních

AudioTrack

Je-li k zařízení připojeno některé z povoleného příslušenství, jsou skrze něj v reálném čase přehrávána získaná zvuková data. Pro tento účel je vhodné využít třídu *AudioTrack*. Obdobně jako inicializace třídy *AudioRecord* i tato třída v konstruktoru očekává parametry definující způsob záznamu zvukových dat, jež se chytáme skrze instanci této třídy přehrát. Nespornou výhodou této třídy je možnost přehrávat syrová PCM data.[13]

Po vytvoření instance této třídy je rozhodnuto na základě příslušenství, zdali budou zvuková data přehrávána či nikoliv. Pokud je podmínka splněna, je zavolána metoda *play()*. Dále je nutné do této třídy zvuková data cyklicky zapisovat. K tomu lze využít **stejný cyklus**, ve kterém jsou zvuková data získávána třídou *AudioRecord*. Získaná zvuková data jsou zapisována do vyrovnávací paměti pomocí metody *write()*, jež byla inicializována při vytvoření instance třídy *AudioTrack*. Zapsaná data jsou dále přehrána skrze připojené příslušenství. Přehrávání je ukončeno zavoláním metody *stop()*. Práce se třídou *AudioTrack* je uvedena ve výpisu 4.13.[13]

Jelikož jsou srdeční ozvy slabě slyšitelné, je hlasitost zařízení automaticky nastavena na maximální úroveň. Ovšem i toto zesílení nemusí být pro dobrou slyšitelnost dostačující. Z toho důvodu byl přidán další stupeň zesílení, jenž přímo multiplikuje amplitudu zvukových dat konkrétní hodnotou navolenou v GUI - v rozsahu 1,0–10,0.

```
// Inicializace
mAudioTrack = new AudioTrack(
    AudioManager.STREAM_MUSIC, // Výstup
    8000, // Vzorkovací frekvence
    AudioFormat.CHANNEL_OUT_MONO, // Nastavení kanálů
    AudioFormat.ENCODING_PCM_16BIT, // Nastavení kódování a bitové hloubky
    BUFFER_SIZE, // Velikost vyrovnávací paměti
    AudioTrack.MODE_STREAM); // Mód přehrávání

// Pole pro čtení dat z vyrovnávací paměti (pro metodu read() třídy AudioRecord)
short rawData[] = new short[BUFFER_SIZE];

// Pole pro zápis modifikovaných dat (pro metodu write() třídy AudioTrack)
short playData[] = new short[BUFFER_SIZE];

// Pokud je připojeno některé z povolených příslušenství
if(isHeadphonesPlugged()) {
    // Nastavení hlasitosti médií na maximum
    mAudioManager.setStreamVolume(AudioManager.STREAM_MUSIC,
        mAudioManager.getStreamMaxVolume(AudioManager.STREAM_MUSIC), 0);

    // Zahájení přehrávání zvukových dat
    mAudioTrack.play();
}

// Zastavení cyklu vyžaduje změnu proměnné mIsRecording na hodnotu false
while (mIsRecording) { // Ukončeno například po kliknutí na tlačítko STOP
    // Čtení zvukových dat z vyrovnávací paměti třídy AudioRecord
    int numRead = mAudioRecord.read(rawData, 0, BUFFER_SIZE);

    // Pokud je třída AudioTrack ve stavu přehrávání
    if (mAudioTrack.getPlayState() == AudioTrack.PLAYSTATE_PLAYING) {
        if (numRead > 0) {
            for (int i = 0; i < numRead; ++i) {
                // Multiplikace amplitudy (mGain - hodnota z GUI)
                playData[i] = (short)Math.min((int)(rawData[i] * mGain),
                    (int)Short.MAX_VALUE);
            }
        }
        // Zápis zvukových dat do vyrovnávací paměti třídy AudioTrack
        mAudioTrack.write(playData, 0, rawData.length);
    }
}

// Ukončení přehrávání a nastavení hlasitosti médií na původní úroveň
mAudioTrack.stop();
mAudioManager.setStreamVolume(AudioManager.STREAM_MUSIC, mOrigionalVolume, 0);
```

Výpis 4.13: Ukázka se třídou *AudioTrack*

4.6.2 Zpracování zvukových dat

Analogový audio signál převedený pomocí lineární PCM na signál digitální je zapisován přímo do souborového systému. Kvalita převodu z analogového signálu na

digitální je dána vzorkovací frekvencí a bitovou hloubkou. Vzorkovací frekvence určuje, kolik vzorků je získáno ze spojitého (analogového) signálu při jeho převodu na signál diskrétní (digitální) za jednu sekundu - tato fáze se nazývá vzorkování. Bitová hloubka poté určuje, v jakém rozsahu lze hodnotu vzorku vyjádřit - tato fáze se nazývá kvantování.[47]

Na základě informací z CDD musí pro záznam syrového audio signálu veškerá zařízení podporovat formát *PCM* o bitové hloubce 16 bitů a alespoň jeden audio kanál - mono. Na výběr je ovšem více vzorkovacích frekvencí - 8 000 Hz, 11 025 Hz, 16 000 Hz a 44 100 Hz. Vzorkování těmito frekvencemi je provedeno přímo konkrétním analogově digitálním převodníkem (ADC - Analog-Digital Converter), jenž zahrnuje i odpovídající antialiasingový filtr. Viz CDD 4.3.[2]

V sekci 3.4.1, je uvedeno, že minimální vzorkovací frekvence FKG signálu je 800 Hz, jinak dojde k porušení vzorkovacího teorému a tím k neobnovitelné ztrátě dat. Na tomto základě lze tedy využít jakoukoliv z výše uvedených vzorkovacích frekvencí. Ovšem pro účely snímání FKG signálu je nejlepší volbou nejnižší vzorkovací frekvence, a to pouze s ohledem na nižší počet vzorků ke zpracování.

Formát PCM a WAV

Jak již bylo zmíněno výše, zvuková data jsou čtena z vyrovnávací paměti vytvořené instancí třídy *AudioRecord*. Ta jsou ukládána ve formátu *PCM*, který je po synchronizaci audio a video záznamu převeden na formát *WAV*. Důvodem je fakt že formát *PCM* neobsahuje hlavičku - jedná se pouze o nekomprimovaný bitový tok zvukových dat. Ovšem bez informací o parametrech získání zvukových dat není možné takovýto záznam přehrát ani zpracovat. Proto je nutné převést formát *PCM* na *WAV*. Formát *WAV* je obal (kontejner) pro uložení bitového toku komprimovaných či nekomprimovaných zvukových dat. Tento formát již obsahuje hlavičku, ve které jsou uvedeny potřebné informace o způsobu kódování, počtu bitů na vzorek, vzorkovací frekvenci, počtu kanálů apod. Díky tomu je možné soubor přehrát či zpracovat na většině zařízení při zachování původních dat ve formátu *PCM*.[10, 47]

4.7 Synchronizace záznamů

Nahrávání audio i video záznamu běží současně - každé ve svém vlákně. Ovšem doba přípravy na nahrávání audio a video záznamu se výrazně liší. Toto zpoždění je nutné velmi přesně korigovat. Odchylka větší než ±50 ms způsobí, že zdravému uživateli (systolický tlak 120 mmHg) bude diagnostikována hypertenze (+50 ms) či hypotenze (-50 ms).

4.7.1 Časové základny

V systému *Android* jsou dostupné tři rozdílné časové základny (hodiny). Tyto časové základy by mezi sebou **neměli být porovnávány**.[54]

První časová základna reprezentována metodou *currentTimeMillis()* třídy *System* je **vyjádření** standardního času a data v jednotkách milisekund. Tento čas by měl být použit pouze ve spojení s časem reálného světa. Například časový údaj 18. 5. 2018 12:30:30 na této časové základně odpovídá hodnotě 1526646630000 ms. Tato časová základna je použita pro pojmenování souborů, které jsou výstupem této aplikace.[54]

Další časové základy již **nevyjadřují** standardní čas. Pomocí těchto časových základen lze měřit pouze uplynulou dobu - hodnoty vrácené těmito časovými základnami jsou smysluplné pouze tehdy, je-li **vypočítán rozdíl mezi dvěma zaznamenanými hodnotami** na tomtéž systému (zařízení). Jednotlivé metody vždy vracejí hodnotu, jež reprezentuje uplynulou dobu od spuštění systému. Rozdíl v následujících časových základnách je primárně určen tím, zdali je měření uplynulého času od spuštění systému zastaveno, pokud systém vstoupí do hlubokého spánku či nikoliv.[54]

Druhá ze tří časových základen, měří uplynulý čas od spuštění systému **bez započítání** času stráveného v hlubokém spánku. Ta je primárně reprezentována metodou *uptimeMillis()* třídy *SystemClock*, jež vrací tuto hodnotu v milisekundách. Tuto časovou základnu využívají i jiné metody pro měření času, jako je například metoda *nanotime()* třídy *System*. Ta totiž vrací velmi přesnou hodnotu času virtuálního stroje, na kterém běží operační systém Android, v nanosekundách.[53, 54]

Poslední, třetí, časová základna měří uplynulý čas od spuštění systému, přičemž je **započítána** i doba strávená v hlubokém spánku. Tato časová základna je primárně reprezentována metodou *elapsedRealtime()* třídy *SystemClock*, jež vrací hodnotu v milisekundách. Případně metodou *elapsedRealtimeNanos()*, která vrací přesnější hodnotu v nanosekundách.[54]

Pochopení výše uvedených časových základen systému *Android* je důležité pro práci s časovými známkami, jejichž hodnoty jsou obvykle vztaženy k některé z časových základen. I v tomto případě ovšem existují výjimky, kdy si konkrétní HW udržuje vlastní časovou základnu.[54]

4.7.2 Synchronizace počátků

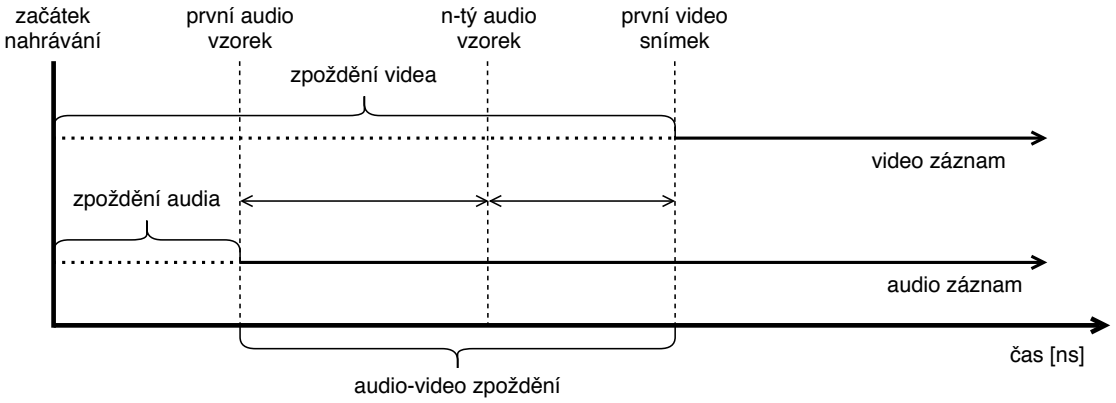
Abychom byli schopni synchronizovat oba záznamy je nutné zaznamenat počátek snímání každého z nich. Princip synchronizace audio a video záznamu je vyobrazen na obrázku 4.6. U videozáznamu je počátkem časová známka prvního zaznamenaného snímku pomocí metody *getTimestamp()* třídy *Image*. U audio záznamu

nejsme bohužel schopni zaznamenat čas akvizice prvního vzorku. Ovšem pomocí metody *getTimestamp()* třídy *AudioRecord* jsme schopni získat čas akvizice n-tého vzorku. Na základě těchto hodnot jsme schopni záznamy vzájemně synchronizovat dle vztahu 4.2.[53]

$$\triangle n \doteq f_{audio} \cdot (t_{video} - t_{audio}(n)) \cdot 10^{-9} + n \tag{4.2}$$

Kde t_{video} odpovídá času, kdy byl získán první snímek videozáznamu, $t_{audio}(n)$ odpovídá času, kdy byl získán n-tý vzorek audiozáznamu, n je pozice audio vzorku, 10^{-9} je převod nanosekund na sekundy, f_{audio} je vzorkovací frekvence audio záznamu a $\triangle n$ udává zpoždění mezi audio a video záznamem ve vzorcích audiozáznamu. Příklad výpočtu na reálných datech získaných v módu ladění.

$$\triangle n \doteq 8\ 000\ Hz \cdot (322\ 486\ 861\ 253\ 416\ ns - \\ - 322\ 486\ 033\ 698\ 339\ ns) \cdot 10^{-9} + 1\ 120 \doteq 7\ 740$$



Obr. 4.6: Synchronizace audio a video záznamu

Obě časové známky jsou vztažené k časové základně měřící uplynulý čas od startu systému včetně času stráveného v hlubokém spánku.

Metoda *getTimestamp()*, pro získání časové známky n-tého vzorku audiozáznamu, s parametrem *AudioTimestamp.TIMEBASE_BOOTTIME* vrací hodnotu na základě metody *elapsedRealtimeNanos()* třídy *SystemClock*.[54]

Zdroj časové známky metody *getTimestamp()* třídy *Image* závisí na zdroji, který obrázek poskytuje. V případě této aplikace je poskytovatelem instance třídy *CameraDevice*, jež reprezentuje konkrétní fotoaparát. Časové známky poskytnuté senzorem fotoaparátu ovšem nemusí být založeny na časové základně, kterou lze porovnávat s

výše uvedenými časovými základnami. Informaci o časové základně senzoru lze zjistit pomocí instance třídy *CameraCharacteristics* konkrétního fotoaparátu (obdobně jako při získávání informace o přítomnosti přisvětlovací diody uvedené v ukázce 4.2) za pomoci parametru *SENSOR_INFO_TIMESTAMP_SOURCE*. Informace nabývá dvou stavů *UNKNOWN* (0) a *REALTIME* (1).[19]

V případě stavu *UNKNOWN* **není možné porovnat** časové známky, kterými byly označeny jednotlivé snímky, s jinými subsystémy - s časovými známkami audiozáznamu. V tomto případě **nelze** audio a videozáznam synchronizovat s dostatečnou přesností, která je vyžadována pro správný výpočet hodnoty krevního tlaku. Tento stav se vyskytuje bohužel **mnohem** častěji, nežli stav *REALTIME*.[19]

Stavu *REALTIME* odpovídá použití stejné časové základny jako v případě metody *elapsedRealtimeNanos()* třídy *SystemClock*. Díky tomu je možné porovnat časové známky mezi oběma záznamy.[19, 54]

4.7.3 Přesnost senzorů

I přes fakt, že senzor fotoaparátu použitého zařízení reprezentuje časovou základu společnou s metodou *elapsedRealtimeNanos()* **nebylo možné signály přesně synchronizovat**.

Problémem se ukázala býti nedostatečná přesnost časových známek produkovaných konkrétními senzory. V CDD verze 7.0 v sekci *7.3. Sensors* je uvedeno následující.[2]

- Všechny senzory **by měly** poskytovat čas akvizice v nanosekundách synchronizovaný s časovou základnou metody *elapsedRealtimeNano()* třídy *SystemClock*. Chyba této synchronizace **by měla být nižší než 100 milisekund**.
- Všechny senzory **musí**, při aktivním záznamu, poskytnout data do stanovené maximální doby dané vztahem 4.3. V případě senzorů s požadovanou minimální dobou je maximální čekací doba dána vztahem 4.4.
- Všechny senzory **musí** poskytnout první vzorek signálu do maximální doby dané vztahem 4.5 od doby aktivace senzorus. Dále je přijatelné, aby tento vzorek měl nulovou přesnost.

$$latence[ms] = 100\ ms + 2 \cdot t \tag{4.3}$$

$$latence[ms] = \quad 5\ ms + 2 \cdot t \tag{4.4}$$

$$latence[ms] = 400\ ms + 2 \cdot t \tag{4.5}$$

Kde *latence* je maximální čekací doba, do které musí být poskytnut vzorek a t je doba potřebná na akvizici tohoto vzorku.[2]

Zpoždění v rozsahu desítek až stovek milisekund není pro synchronizaci přípustné. Z toho důvodu byla snaha toto zpoždění korigovat. Prvotním pokusem bylo stanovení konstanty, pomocí níž by bylo zpoždění odstraněno. Ovšem při měření zpoždění mezi jednotlivými záznamy bylo zjištěno, že hodnota je příliš **variabilní**. Rozsah dodatečného zpoždění (po synchronizaci počátků - viz 4.7.2) se pohyboval v rozsahu 200–600 ms, a proto nebylo možné toto zpoždění korigovat za pomoci konstanty. Byly provedeny i další pokusy o korekci zpoždění, jenž měřily uplynulou dobu před i při akvizici v různých částech aplikace. Bohužel **nebyl nalezen** žádný způsob, jak signály vzájemně synchronizovat s přesností v rámci jednotek milisekund či méně.

Z toho důvodu je nutné vytvořit v obou signálech synchronizační značky za pomoci vnějšího podnětu. Tímto podnětem může být například poklepání prstu přes objektiv fotoaparátu, což způsobí jeho krátkodobé zakrytí a zároveň generování vibrací, jež jsou zaznamenány mikrofonem. Současně tak vznikne v obou signálech výkyv, dle kterého lze signály synchronizovat bez ohledu na časové známky. Tato metoda synchronizace je převzata z [26].

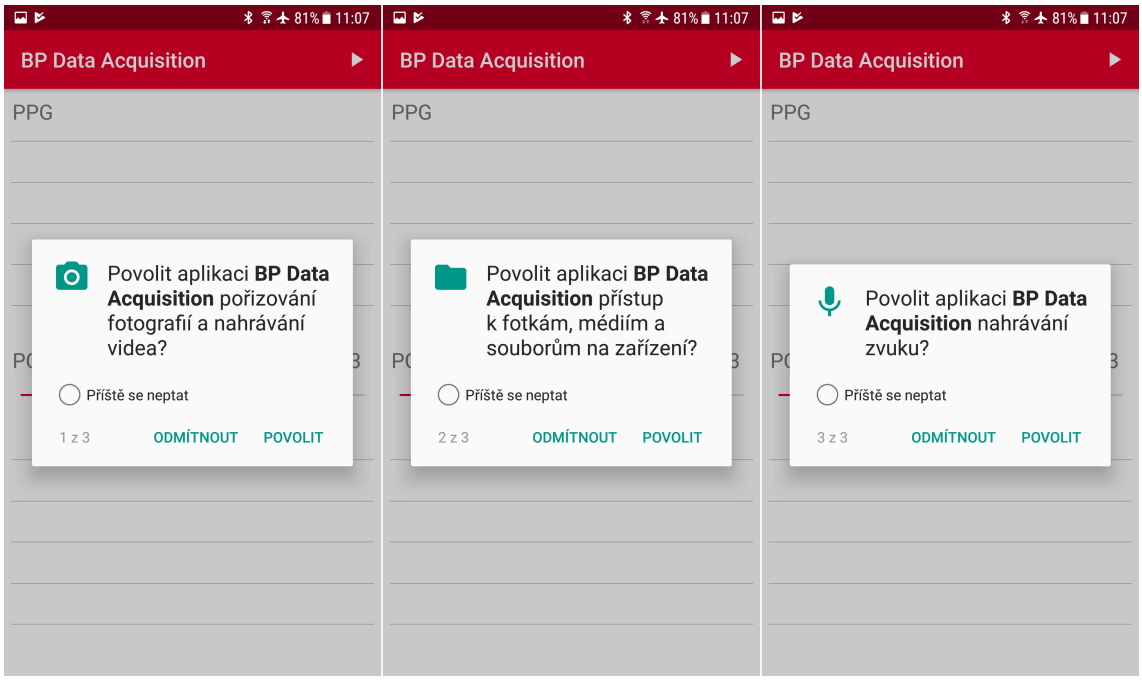
4.8 Hlavní spouštěcí aktivita

Po spuštění aplikace je otevřena hlavní spouštěcí aktivita nesoucí název *MainActivity*. Aplikace obsahuje pouze tuto aktivitu - pouze tuto obrazovku. Účelem této aktivity je zprostředkovat GUI pro interakci s uživatelem a vytvořit funkční celek s výše uvedenými třídami.

Před použitím aplikace je nutné zkontrolovat, zdali má aplikace přidělena dostatečná oprávnění. Pokud některé z oprávnění chybí, je uživatel požádán o jeho přidělení. Tato skutečnost je vyobrazena na obrázku 4.7. V případě neudělení některého z oprávnění je aplikace bezpodmínečně ukončena.

GUI tvoří základní komponenty, vyjma grafů, nabízené systémem Android. Vykreslení grafů je zajištěno pomocí knihovny *GraphView*. Jedná se o open source knihovnu umožňující vytvářet bodové, spojnicové či sloupcové grafy. Hlavní výhodou je ovšem možnost vytvoření živého grafu - data je možné měnit za běhu, takže lze vykreslovat změny v reálném čase. U většiny vyzkoušených knihoven nebyla tato možnost dostupná - docházelo k překreslování celého grafu.

Snímek obrazovky vytvořený po spuštění aplikace a snímek obrazovky po ukončení nahrávání je vyobrazen na obrázku 4.8. GUI obsahuje pouze prvky nutné k poskytnutí zpětné vazby a řízení akvizice dat. V pravém horním rohu se nachází tlačítko umožňující zahájení a ukončení akvizice dat. Toto místo je ideální pro ovládání aplikace pouze jednou rukou. V horní polovině obrazovky jsou vykreslována již zpracovaná obrazová data třídy *ImageReader*. Jedná se o průměrnou jasovou hodnotu



Obr. 4.7: Žádost o přidělení oprávnění po spuštění aplikace

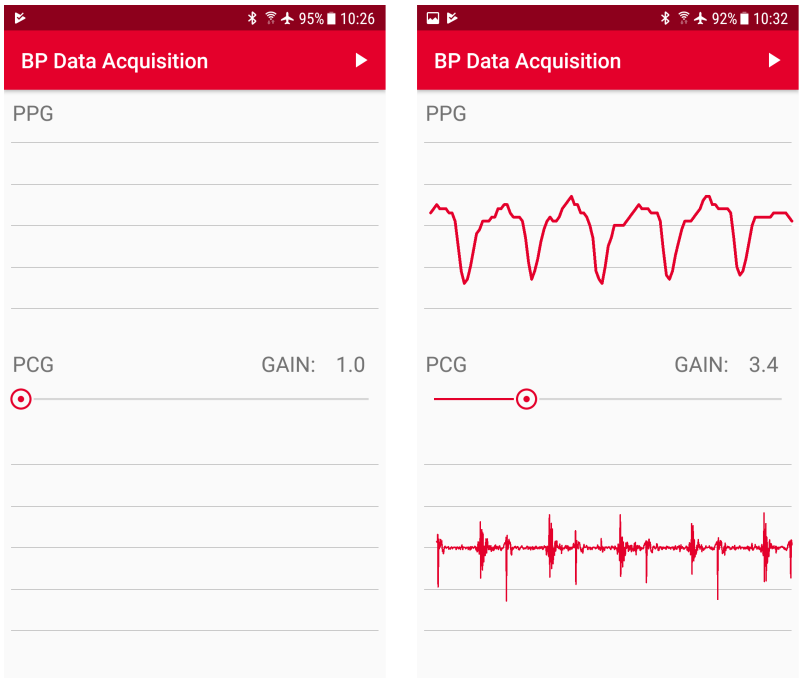
červené složky snímku pořízeného z konečku prstu - PPG záznam. V dolní polovině obrazovky jsou vykreslována zvuková data třídy *AudioRecord*. Jedná se o záznam srdečních ozev nasnímaných pomocí mikrofonu přiloženého na kůži v mitrální oblasti - FKG záznam. Jelikož se vykreslování 8 000 vzorků každou sekundu (vzorkovací frekvence audio signálu) ukázalo jako nepraktické (zamrzání GUI) je vykreslován každý dvaatřicátý vzorek. Nad grafem se nachází posuvník, jenž určuje a umožňuje dodatečné zesílení přehrávaného audio signálu skrze připojené příslušenství.

Po ukončení akvizice dat je vypočítáno zpoždění video signálu, které je korigováno. Výstupem měření jsou dva soubory - audio ve formátu **.wav* a video ve formátu **.mp4*. Oba soubory se nacházejí v interní paměti telefonu ve složce *\Android\data\cz.vutbr.bpdataacquisition\files*. Tyto soubory je nutné dále zpracovat v prostřední *MATLAB*.

4.9 Testování aplikace

Aplikace byla po čas vývoje intenzivně testována a modifikována. Při testování byly ovšem odhaleny i nedostatky, které **nebylo možné** kompenzovat. Tyto nedostatky se vztahují k metodě měření krevního tlaku pomocí mobilního telefonu, operačnímu systému *Android*, konkrétnímu zařízení či kombinaci všech uvedených.

Prvním z nedostatků je energetická a výpočetní náročnost. Zpracování zvukových



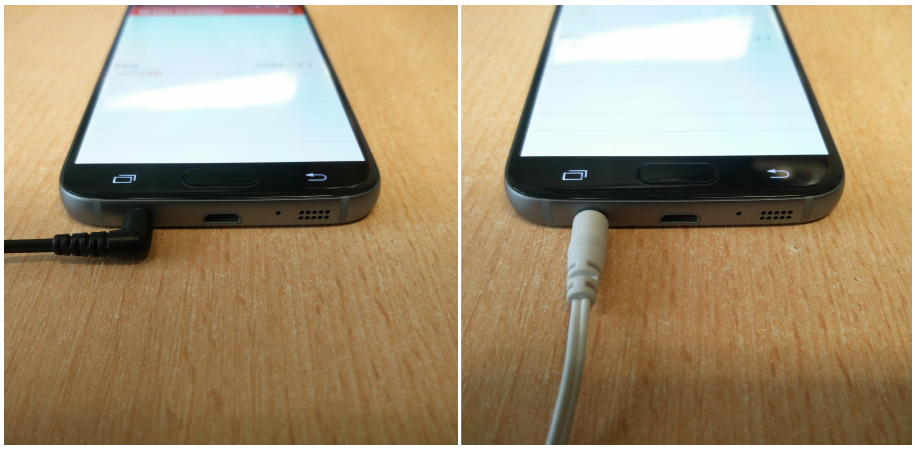
Obr. 4.8: GUI aplikace - po spuštění (vlevo), po ukončení nahrávání (vpravo)

a obrazových dat značně vytěžuje všechny dostupné výpočetní jednotky. Zejména zpracování obrazových dat (převod formátu *YUV_420_888*) vyžaduje kooperaci CPU a GPU. Takovéto zatížení výpočetních jednotek způsobí vyšší spotřebu elektrické energie a zahřívání zařízení, díky čemuž se uživateli potí dlaně, což snižuje komfort této metody. Komfort dále snižuje nepříjemně se zahřívající LED, na které je nutné mít při měření přiložený koneček prstu. Při delší akvizici dat (více než 3 minuty) či více po sobě jdoucích měřeních se LED zahřeje natolik, že je nutné měření přerušit, jinak by mohlo dojít k **popálení přiloženého prstu**. LED je také významným konzumentem elektrické energie. Aplikace za 30 minut běhu, ze kterých bylo 20 minut nahráváno (více krátkých záznamů) spotřebovala 325 *mAh*. Použité zařízení má kapacitu baterie 3 000 mAh. Bylo tedy spotřebováno přibližně 11 % elektrické energie uložené v baterii. Informace o spotřebě elektrické energie jednotlivých aplikací či HW lze obvykle zjistit v nastavení zařízení, sekce údržba, baterie, využití baterie.

Druhý nedostatek se vztahuje ke konkrétnímu zařízení. Použité zařízení, *Samsung Galaxy S7*, má na spodní bradě tři tlačítka - *domů*, *zpět* a *přepnout*. Tlačítka *zpět* a *přepnout* jsou dotyková (kapacitní). Tlačítko *domů* je mechanické. Tato skutečnost je vyobrazena na obrázku 4.9. Dotyková tlačítka jsou velmi citlivá, a tak při měření občas dochází k dotyku kůže s některým z těchto tlačítek. To způsobí vyvolání události, jež je k tlačítku přiřazena. Z pohledu vývoje aplikace, lze modifikovat

pouze událost tlačítka *zpět* - zabránit ukončení aplikace. Události tlačítek *domů* a *přepnout* **nelze v rámci aplikace obsloužit**, a to z důvodu, aby se zabránilo škodlivému SW v blokování zařízení (aplikace, kterou nelze ukončit) - tato funkce je **blokována systémem**. V případě dotyku tlačítka *přepnout* tedy nelze jeho událost modifikovat. Ovšem dotyk tohoto tlačítka způsobí otevření obrazovky s aktuálně spuštěnými aplikacemi. Jelikož uživatel opouští aktuální aktivitu (obrazovku), je nutné ji pozastavit. Opuštění aktivity způsobí zavolání metody *onPause()*, ve které aktivitu opět obnovíme - přesnuneme proces do popředí. Této funkčnosti je využito pouze při nahrávaní. Pro událost tlačítka *domů* bohužel neexistuje žádné přijatelné řešení. Naštěstí k mechanickým tlačítkům se tento nedostatek nevztahuje, a proto v našem případě byl tento problém vyřešen. Toto ovšem nemusí platit pro jiná zařízení.[55]

Třetím nedostatkem je nevhodné umístění vstupu pro připojení drátových sluchátek. V případě použitého zařízení se vstup pro připojení sluchátek s *3,5 mm Jack* konektorem nachází ve spodní bradě telefonu. Na tomto místě se taktéž nachází hlavní mikrofon. Takovéto umístění mikrofonu je pro mobilní telefony typické. Při měření je nutné mikrofon přitlačit na kůži v mitrální oblasti. Pokud jsou ovšem k zařízení připojena drátová sluchátka s **přímým konektorem** není to možné. S drátovými sluchátky se **zahnutým konektorem** lze měřit, ale komfort uživatele je snížen. Porovnání těchto dvou typů konektorů je vyobrazeno na obrázku 4.9. Kvůli tomuto nedostatku a novodobým zařízením, kterým byl tento vstup odejmutý, byla implementována možnost připojení zvukového příslušenství přes bluetooth.



Obr. 4.9: Připojení sluchátek přes *3,5 mm Jack* konektor. Zahnutý typ konektoru (vlevo), přímý typ konektoru (vpravo). Tlačítka zařízení (oba obrázky) - dotykové tlačítko *přepnout* (levé tlačítko), mechanické tlačítko *domů* (prostřední tlačítko), dotykové tlačítko *zpět* (pravé tlačítko).

5 Zpracování dat

Záznamy získané pomocí aplikace, popsané v předchozí kapitole, jsou zpracovány v prostředí *MATLAB*. Výstupem aplikace jsou dva soubory. První soubor je videozáznam ve formátu *MP4*, ze kterého bude získán PPG signál. Druhý soubor je audiozáznam ve formátu *WAV*, ze kterého bude získán FKG signál. Oba soubory si v hlavičce nesou informace o jejich akvizici - tzv. metadata, proto není potřebné zadávat informace manuálně.

5.1 Předzpracování signálů

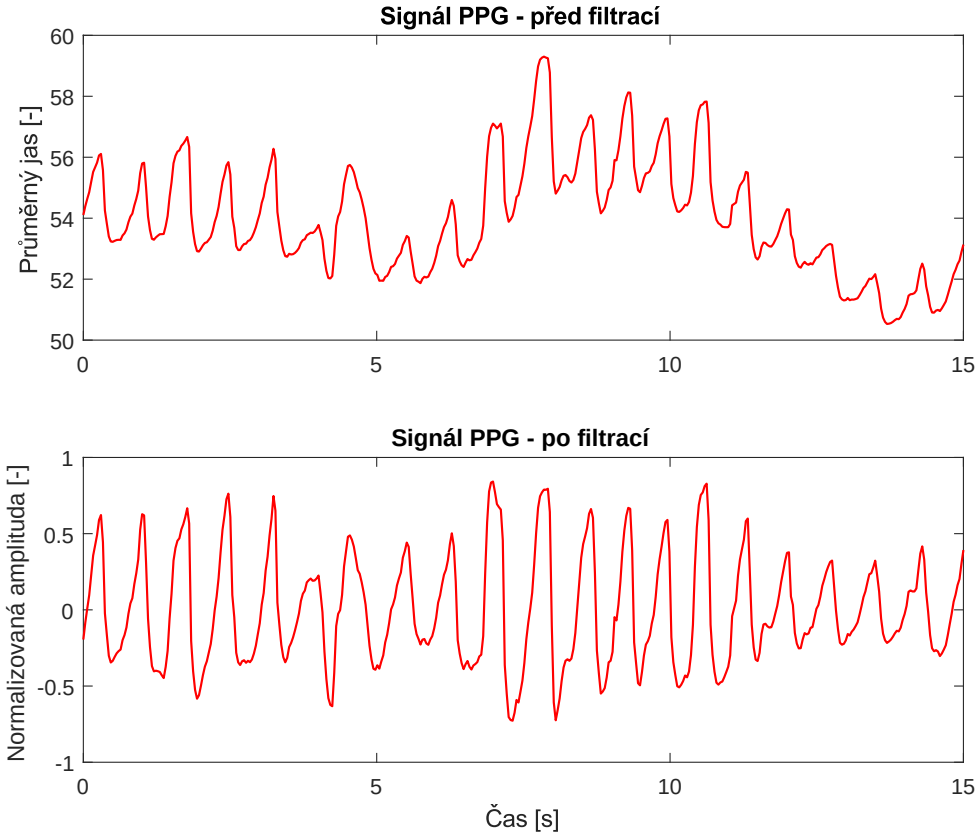
Před výpočtem hodnoty krevního tlaku je nutné nejprve získat příslušné signály v požadované kvalitě tak, aby bylo možné naleznout zájmové body v těchto signálech. Těmito zájmovými body jsou obě srdeční ozvy z FKG signálu a maximum plusní vlny v PPG signálu.

5.1.1 PPG signál

PPG signál je extrahován z videozáznamu pomocí funkce *VideoReader()*, jejímž vstupem je cesta k souboru. Tato funkce vrací objekt, který poskytuje informace o akvizici videozáznamu (vzorkovací frekvence, rozměry videozáznamu apod.). Déle umožňuje číst video po jednotlivých snímcích. Tyto snímky jsou automaticky převedeny z formátu *YUV* na formát *RGB*, což značně usnadňuje jejich zpracování. Každý snímek je reprezentován třídimenzionální maticí, kde počet řádků a sloupců odpovídá rozměru snímku a dimenze reprezentuje jednotlivé složky barevného modelu *RGB*.

Pro získání PPG signálu je důležitá pouze první dimenze, jež reprezentuje červenou složku barevného modelu *RGB*. Jeden vzorek PPG signálu je poté získán jako průměr jasových hodnot červené složky konkrétního snímku. Ostatní barevné složky neobsahují informaci o průchodu pulsní vlny konečkem prstu a jsou tedy redundantní.

Aby bylo možné PPG signál dobře zpracovat, je nutné ze signálu odstranit pohybové artefakty. Z tohoto důvodu je signál filtrován filtrem typu horní propust s mezní mezní frekvencí 0,8 Hz. Tato mezní frekvence byla stanovena empiricky, jelikož mezní frekvence 0,5 Hz, uvedená v [26], nebyla dostačující. Odstranění pohybových artefaktů je důležité pro následnou detekci maxim jednotlivých pulsních vln. Signál před a po filtraci je vyobrazen na obrázku 5.1.



Obr. 5.1: Filtrace PPG signálu filtrem typu horní propust s mezní frekvencí 0,8 Hz pro odstranění pohybových artefaktů.

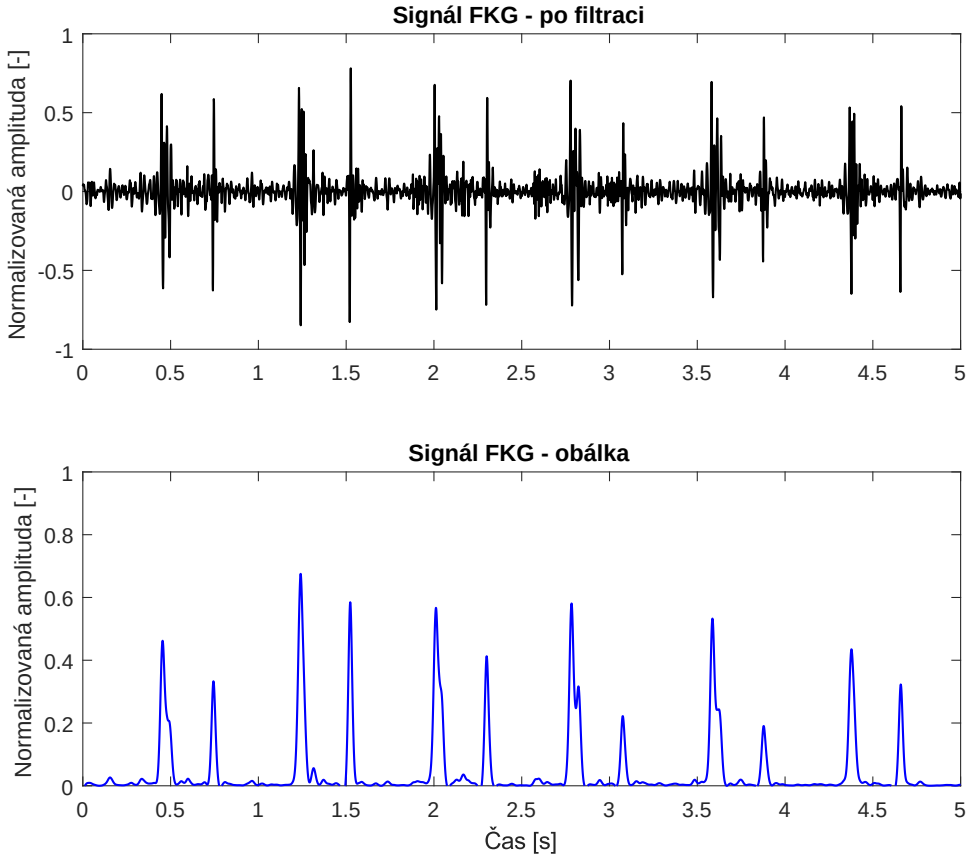
5.1.2 FKG signál

FKG signál je extrahován z audiozáznamu za pomocí funkce *audioread()*, jejímž vstupem je cesta k souboru. Výstup této funkce je pole vzorků jednotlivých kanálů a vzorkovací frekvence.

Audiozáznam získaný prostřednictvím aplikace, popsané v předchozí kapitole, obsahuje pouze jeden kanál. Vzorkovací frekvence extrahovaného audiozáznamu je 8 kHz. To je nejnižší možná vzorkovací frekvence, jež může být zaznamenána na operačním systému *Android* verze 7.0. I přes to, že se jedná o nejnižší vzorkovací frekvenci, je pro zpracování FKG signálu stále redundantní. Z toho důvodu je vhodné snížit vzorkovací frekvenci z 8 kHz na 1 kHz.

Dalším krokem je odstranění nežádoucích složek signálu. Toho je docíleno použitím IIR filtru typu pásmová propust s dolní mezní frekvencí 20 Hz a horní mezní frekvencí 250 Hz. Krátký úsek FKG signálu po filtraci tímto filtrem je vyobrazen na obrázku 5.2. [26]

Po této filtraci je signál normalizován v rozsahu od -1 do 1. Dále je získána energie signálu umocněním každého vzorku druhou mocninou. Posledním krokem předzpracování je získání obálky signálu pomocí IIR filtru typu dolní propust s mezní frekvencí 20 Hz. Výsledná obálka FKG signálu je vyobrazena na obrázku 5.2.[26]



Obr. 5.2: Signál FKG po filtraci a obálka tohoto signálu

5.2 Synchronizace záznamů

I přes to, že oba záznamy byly již synchronizovány v aplikaci, je zpoždění mezi nimi stále **výrazné**. Důvodem je nepřesnost senzorů, kterými jsou poskytnuty časové známky, jejichž časové základny si neodpovídají. Z toho důvodu nelze korigovat zpoždění s přesností v rámci jednotek milisekund či méně. Tato problematika je podrobně rozebrána v sekci 4.7.3. V této sekci je také popsán postup pro vytvoření synchronizačních značek v obou záznamech poklepu prstu, které jsou pro tuto synchronizaci důležité.

Poté, co byly signály předzpracovány, je uživatel nucen vybrat úsek signálu, ve kterém se nalézá synchronizační značka. V tomto úseku PPG signálu a obálky FKG signálu jsou hledány indexy maximálních hodnot. Na základě indexů je zjištěna časová hodnota vzorků, pomocí které je vypočítáno zpoždění. Z FKG signálu a jeho obálky je poté odstraněn příslušný počet vzorků. Díky tomu jsou signály velmi přesně synchronizovány (v jednotkách milisekund). Tento postup synchronizace se ukázal jako velmi přesný. Ukázka synchronizace PPG a FKG záznamu na základě synchronizačních značek je uvedena ve výpise 5.1.

Pokud jsou oba záznamy přesně synchronizovány, lze začít hledat jednotlivé zájmové body, na základě kterých budou stanoveny parametry PTT a ET.

```
% Hodnoty časů pro jednotlivé vzorky (osa x)
vt = linspace(0, length(PPG) / videoFrameRate, length(PPG));
et = linspace(0, length(ENV) / audioSampleRate, length(ENV));

% Úsek signálu se synchronizační značkou
ppgSyncArea = PPG((syncFrom *  videoFrameRate) : (syncTo *  videoFrameRate));
envSyncArea = ENV((syncFrom * audioSampleRate) : (syncTo * audioSampleRate));

% Nalezení indexů maximální hodnot
[~, ppgSyncPos] = max(ppgSyncArea); % PPG signál
[~, envSyncPos] = max(envSyncArea); % Obálka FKG signálu

% Přepočet indexů z úseku se synchronizační značkou na idexy v celém sigálu
ppgSyncIdx = syncFrom * videoFrameRate + ppgSyncPos - 1;
envSyncIdx = syncFrom * audioSampleRate + envSyncPos - 1;

% Zjištění časové hodnoty pro konkrétní indexy
ppgSyncTime = vt(ppgSyncIdx);
envSyncTime = et(envSyncIdx);

% Výpočet zpoždění
delay = round((envSyncTime - ppgSyncTime) * audioSampleRate);

% Odstranění počtu vzorků, jenž odpovídající zpoždění
PCG = PCG(delay + 1 : end);
ENV = ENV(delay + 1 : end);
```

Výpis 5.1: Výpočet zpoždění a synchronizace PPG a FKG signálu

5.3 Detekce vrcholů

Před samotnou detekcí vrcholů je nutné zvolit úsek záznamu, ve kterém se nacházejí zájmové body. Uživatel je automaticky vyzván, aby tento úsek v grafu předzpracovaných signálů vytyčil.

5.3.1 Pulsní vlna

V PPG signálu jsou hledány vrcholy pulsních vln. Vektor pozic vrcholů je nalezen pomocí funkce *findpeaks()*. Tu lze optimalizovat zadáním parametru minimální výšky vrcholu – *MinPeakHeight* a parametru minimální vzdálenosti vrcholů – *MinPeakDistance*. Minimální výška vrcholu byla empiricky stanovena jako 25 % maxima z PPG signálu. Aby nedocházelo k detekci dvou a více vrcholů v rámci jedné pulsní vlny, je nastavena minimální vzdálenost mezi vrcholy. Ta je dána dobou trvání jednoho srdečního cyklu pro maximální tepovou frekvenci, kterou lze nastavit.

Detekce vrcholů v PPG signálu pomocí funkce *findpeaks()* je **velmi přesná**. Výhodou tohoto postupu je detekce i vrcholů s nižší amplitudou, které by při prahové detekci byly ignorovány. Výrazný rozdíl amplitudy jednotlivých pulsových vln v PPG signálu je možné vidět na obrázku 5.1.

5.3.2 První a druhá srdeční ozva

Zájmovými body FKG signálu jsou první a druhá srdeční ozva. Při detekci S1 je využito faktu, že se vrchol této srdeční ozvy musí nacházet před vrcholem korespondující pulsní vlny v PPG signálu a zároveň se musí nacházet za vrcholem předchozí pulsní vlny. V oblasti mezi dvěma S1 se poté nachází S2.

První verze algoritmu pracovala na principu hledání maxima v obálce FKG signálu právě mezi korespondujícím a předchozím vrcholem pulsní vlny v PPG signálu. Tento algoritmus se ovšem ukázal jako nevyhovující. Mezi měřenými subjekty se totiž vykytovaly jedinci, u nichž byla amplituda S2 vyšší než amplituda S1. To způsobilo nekorektní detekci S1 a následně S2.

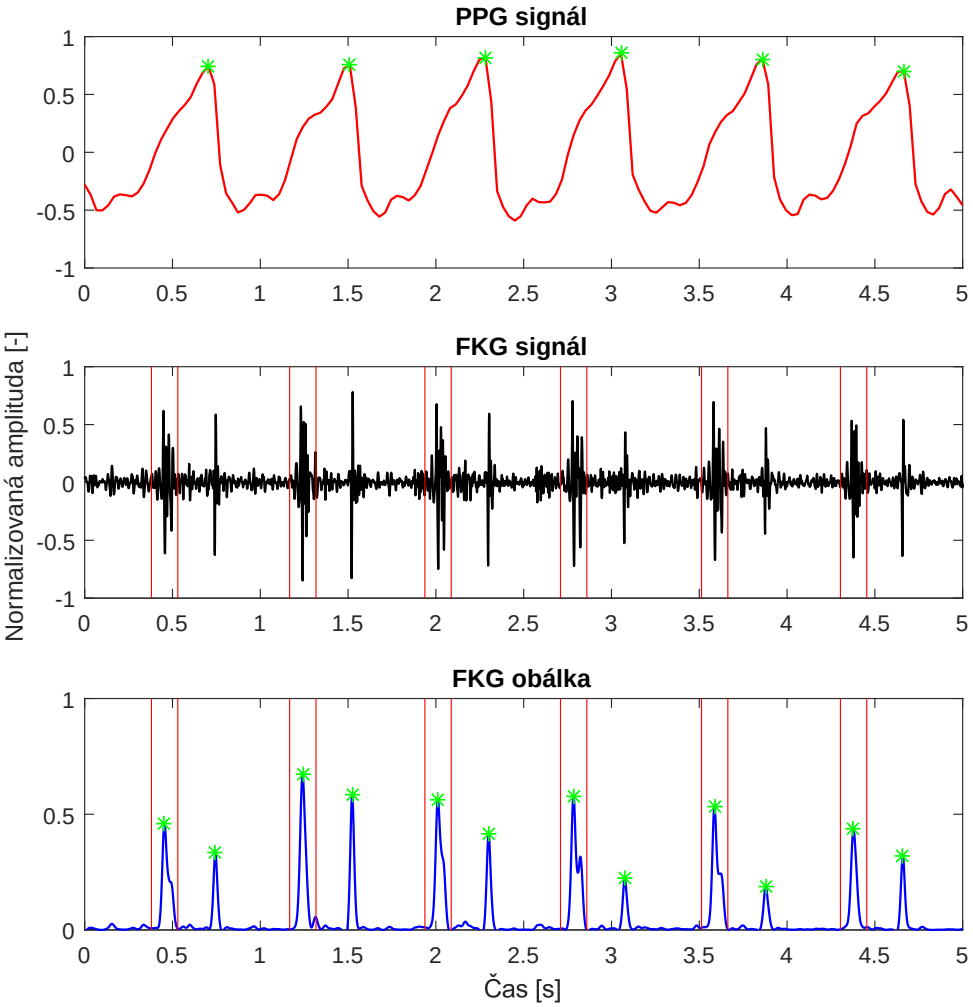
Druhá verze algoritmu již předpokládá určitý rozsah PTT. Jinými slovy, pokud bude S1 příliš blízko korespondujícímu vrcholu pulsní vlny, bude vypočítaný systolický tlak příliš vysoký. V opačném případě bude vypočítaný systolický tlak příliš nízký. Rozsah minimálního a maximálního systolického tlaku, ve kterém algoritmus pracuje, byl empiricky stanoven v rozmezí 80–180 mmHg. Minimální a maximální vzdálenost PTT pak lze vypočítat dle vztahu 5.1, kde $p_{S_{min}}$ a $p_{S_{max}}$ udávají rozsah minimálního a maximální tlaku, q je konstanta z rovnice přímky ve směrnicovém tvaru 3.3 a hodnota $-0,425$ je směrnice k této přímky - viz 3.3.1.

$$\begin{aligned} PTT_{min}\ [ms] &= -\frac{p_{S_{max}} - q}{0,425} \\ PTT_{max}\ [ms] &= -\frac{p_{S_{min}} - q}{0,425} \end{aligned} \tag{5.1}$$

Vrchol S1 je pak hledán před vrcholem pulsní vlny ve vzdálenosti větší, než PTT_{min} a zároveň menší, než PTT_{max}. Díky tomuto postupu již není vyžadováno

aby S1 měla větší amplitudu než S2, což umožňuje snímat FKG signál i z jiných oblastí vyobrazených na obrázku 3.3, než pouze z oblasti nad mitrální chlopní.

Aby bylo možné detekovat S2 mezi dvěma S1, je nutné S1 patřičně ohraničit. Doba trvání S1 je ze všech srdečních ozev nejdelší. Při ohraničení S1 vycházíme z fyziologické doby trvání této ozvy, jež může nabývat až 150 ms. Na základě těchto informací je před a za vrcholem S1 definována hranice ve vzdálenosti 75 ms od vrcholu. V oblasti za hranicí za S1, korespondující s S2, a přední hranicí následující S1, je nalezena S2 jako maximum v této oblasti.[30]



Obr. 5.3: Detekce zájmových bodů v PPG a FKG signálu (zelené body). Červené ohraničení v FKG signálu a obálce FKG signálu značí S1.

Korektně provedená detekce zájmových bodů je vyobrazena na obrázku 5.3. Po detekci zájmových bodů již lze bez větších obtížích stanovit hodnotu systolického a diastolického tlaku dle postupu uvedeného v sekci 3.3. Ukázka stanovení hodnoty krevního tlaku je uvedena ve výpisu 5.2.

```
% Výpočet tepové frekvence
HR = ((length(ppgPeaks)+1) * 60) / (signalTo-signalFrom);

% Výpočet PTT
PTT = ((peaks - S1) / audioSampleRate) * 1000;

% Výpočet systolického tlaku
Ps = -0.425 * PTT + q;

% Výpočet doby vypuzovací fáze srdečního cyklu
ET = ((S2 - S1) / audioSampleRate) * 1000;

% Výpočet povrchu těla
BSA = 0.007184 * user.weight^(0.425) * user.height^(0.725);

% Výpočet systolického objemu
SV = -6.6+ 0.25 * (ET - 35) - 0.62 * HR + 40.4 * BSA - 0.51 * user.age;

% Výpočet pulsového tlaku
Pp = SV ./ ((0.013 * user.weight -0.007 * user.age - 0.004 * HR) + 1.307);

% Výpočet diastolického tlaku
Pd = Ps - Pp;
```

Výpis 5.2: Výpočet hodnoty krevního tlaku

6 Vyhodnocení dat

I přes to, že aplikace, vytvořená speciálně pro akvizici dat, signifikantně zvýšila úspěšnost získávání záznamů z chytrého telefonu, bylo získáno pouze 15 relevantních záznamů od čtyř uživatelů. Více než 85 % záznamů bylo odstraněno z důvodu **nedokonalé synchronizace**, na kterou je tato metoda příliš citlivá. Informace o měřených uživatelích jsou uvedeny v tabulce 6.1.

Tab. 6.1: Informace o vyšetřovaných uživatelích

Uživatel č.	Hmotnost [kg]	Výška [cm]	Věk	BSA [m^2]
1	94	179	24	2,13
2	87	182	24	2,09
3	93	171	24	2,05
4	83	168	24	1,93

Každý uživatel podstoupil několik měření. Výsledky těchto měření jsou uvedeny v tabulce 6.2. Při každém měření byla zaznamenána i referenční hodnota krevního tlaku pomocí auskulatační metody.

Tab. 6.2: Výsledky jednotlivých měření

Uživatel č.	$p_{S_{ref}}$ [mmHg]	p_S [mmHg]	$p_{D_{ref}}$ [mmHg]	p_D [mmHg]
1	144	140	88	95
1	144	149	88	115
1	146	145	90	105
1	144	143	88	106
1	140	141	88	106
1	132	134	80	95
2	122	119	70	78
2	120	130	72	89
2	124	131	72	89
2	122	116	72	71
3	142	135	92	95
3	140	140	90	98
3	144	159	90	116
4	112	106	68	69
4	120	111	70	72

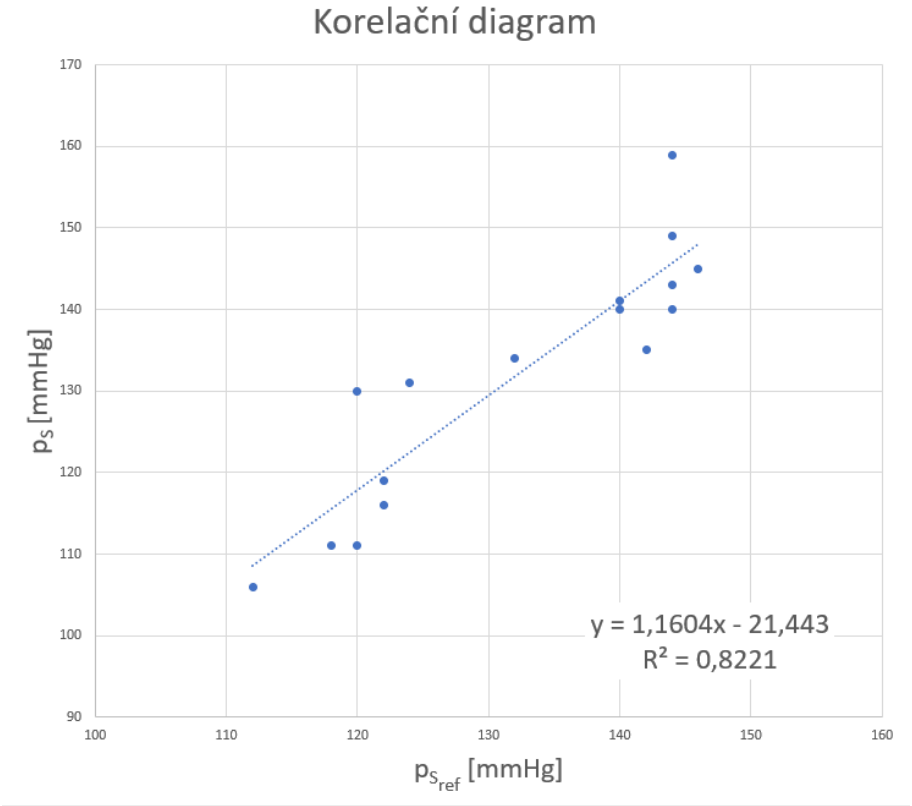
Pří výpočtu hodnoty krevního tlaku, pomocí metody šíření pulsní vlny, nedostáváme pouze jedinou hodnotu. Hodnota systolického i diastolického tlaku je vypoč-

tena při každém srdečním cyklu. Výsledný, vypočítaný, tlak uvedený v tabulce 6.2 je tedy medián všech naměřených hodnot v rámci jednoho záznamu.

Na hladině významnosti $\alpha = 0,01$ jsme otestovali, zdali se hodnoty naměřené auskultační metodou výrazně liší (H_1) od hodnot vypočítaných či nikoliv (H_0). Jelikož se jedná o párová data, byl použit Studentův párový t-test.

Výsledná p hodnota párového t-testu pro hodnoty **diastolického tlaku** je $0,00006627$. Jelikož je p hodnota podstatně menší než $0,01$ zamítáme H_0 a přijímáme H_1. Hodnoty diastolického tlaku naměřené auskulatační metodou se statisticky **výrazně liší** od hodnot vypočítaných.

Výsledná p hodnota párového t-testu pro hodnoty **systolického tlaku** je $0,88399358$. Jelikož je p hodnota podstatně větší než $0,01$ přijímáme H_0 a zamítáme H_1. Hodnoty systolického tlaku naměřené auskulatační metodou se statisticky **neliší** od hodnot vypočítaných. Výsledek párového t-testu pro hodnoty **systolického tlaku** je taktéž podpořen korelačním koeficientem o hodnotě 0,9067.



Obr. 6.1: Korelační diagram mezi $p_{S_{ref}}$ a p_S

7 Závěr

V úvodní části práce je definován pojem krevní tlak. Je popsán jeho vznik a vztah k oběhové soustavě. Dále jsou uvedeny fyziologické hodnoty krevního tlaku a jejich rozsah. Velmi podrobně jsou rozebrány metody neinvazivního měření krevního tlaku.

Další kapitola je rešerše zaměřená na operační systém Android. Seznámení se s tímto operačním systémem a vývojem aplikací je velmi důležité pro vytvoření vlastní aplikace určené k akvizici dat pro výpočet hodnoty krevního tlaku. V rámci této kapitoly je také popsána historie vývoje, jednotlivé verze, architektura a vývojová prostředí.

V třetí kapitole jsou rozebrány nové způsoby měření krevního tlaku, zejména pomocí senzorů chytrého telefonu. Jedná se především o experimentální metody, které nejsou v běžné praxi prozatím využívány. Hlavní metodou této kapitoly je metoda výpočtu hodnoty krevního tlaku na základě výpočtu doby přenosu pulsu (PTT) mezi dvěma místy v arteriálním řečišti. Toho je docíleno snímáním prvního místa pomocí mikrofonu a druhého pomocí fotoaparátu s přisvětloací diodou. První místo náleží oblasti srdce a jeho snímáním tedy získáváme FKG signál. Druhým místem je koneček prstu, jenž je prosvěcován pomocí LED. Část emitovaného světla je rozptýlena, část odražena a část absorbována živou tkání. Díky tomu je možné zachytit průchod pulsní vlny konečkem prstu na základě změny intenzity světla dopadajícího na snímač fotoaparátu. Tím je získán PPG signál. Ovšem snímání těchto signálů bez zpětné vazby je značně náročné.

Z důvodu problematického snímání PPG a FKG signál byla vytvořena aplikace na platformě *Android*. Primární účele této aplikace je poskytnout uživateli zpětnou vazbu o zaznamenávaných signálech. Zpětná vazba PPG signálu je vyobrazena jako graf na displeji telefonu. Zpětná vazba FKG signálu je v audio-vizuální podobě. Uživateli je zobrazen graf na displeji telefonu a zároveň jsou přehrávány zvuky srdečních ozev ve sluchátkách připojených přes Bluetooth či audio konektor. Díky tomu uživatel umístí senzory na správná místa vhodná ke snímání. Tato zpětná vazba signifikantně zvýšila úspěšnost akvizice dat. Aby bylo možné tuto zpětnou vazbu poskytnou, je nutné snímané signály zpracovat. Aplikace zaznamenává zvuková a obrazová data, která jsou po ukončení akvizice k dispozici. Záznamy získané prostřednictvím této aplikace jsou dále zpracovány a vyhodnoceny v prostředí *Matlab*.

Při vývoji aplikace se bohužel objevily problémy se synchronizací signálů, které se nepodařilo vyřešit do takové míry, aby je bylo možné zanedbat. Použitá metoda, pro stanovení hodnoty krevního tlaku, je totiž na této synchronizaci značně závislá. Odchylka větší než ± 50 ms způsobí, že zdravému uživateli bude diagnostikována hypertenze či hypotenze.

Zdrojem zpoždění se ukázala býti časová nepřesnost senzorů. U časových známek,

které jsou použité k synchronizaci je povolená odchylka 100 ms od časové základny, ke které jsou přidruženy. Dále je zde latence po spuštění senzoru, která může nabývat více než 400 ms (rovnice 4.5). V neposlední řadě je zde tolerované zpoždění 100 ms při akvizici dat. Všechna výše uvedená zpoždění jsou variabilní a proto je nelze korigovat.

Vytvoření synchronizační značky, v obou signálech, pomocí vnějšího podmětu (poklepání na fotoaparát - generuje zvuk i změnu jasu) a její následná synchronizace se ukázala jako nejvíce přesná (oproti vyzkoušeným metodám), ale ne dostatečně pro potřeby algoritmu na výpočet hodnoty krevního tlaku.

I přes tyto problémy se ale podařilo naměřit alespoň 15 záznamů u čtyřech různých osob, které byly synchronizovány s dostatečnou přesností. Tyto záznamy byly vyhodnoceny pomocí skriptu v prostředí Matlab, jehož výstupem je hodnota krevního tlaku. Při porovnání s referenčními hodnotami, získanými pomocí auskulatační metody, byla u hodnot systolického tlaku zjištěna statisticky významná shoda. Hodnoty diastolického tlaku se naopak od referenčních statistiky výrazně liší.

Literatura

[1] ALLEN, Grant. *Android 4: průvodce programováním mobilních aplikací.* Brno: Computer Press, 2013. ISBN 978-80-251-3782-6.

[2] Android Compatibility Definition Document. *Android Open Source Project* [online]. 2017 [cit. 2018-05-10]. Dostupné z: <https://source.android.com/compatibility/cdd>.

[3] Android hardware camera2. *Android Developers* [online]. 2018 [cit. 2018-05-01]. Dostupné z: <developer.android.com/reference/android/hardware/camera2/package-summary>.

[4] Android, the world's most popular mobile platform. *Android Developers* [online]. [cit. 2017-11-22]. Dostupné z: <https://developer.android.com/about/android.html>.

[5] Android Studio. *Zdroják* [online]. 2013 [cit. 2017-11-27]. Dostupné z: <https://www.zdrojak.cz/clanky/android-studio-nove-vyvojove-prostredi/>.

[6] Android HdrViewfinder Sample. *GitHub - Google Samples* [online]. 2018 [cit. 2018-05-11]. Dostupné z: <https://github.com/googlesamples/android-HdrViewfinder>.

[7] An Introduction to Renderscript. *Microsoft Docs* [online]. 2018 [cit. 2018-05-11]. Dostupné z: <https://docs.microsoft.com/en-us/xamarin/android/platform/renderscript>.

[8] Application Fundamentals. *Android Developers* [online]. 2017 [cit. 2017-11-27]. Dostupné z: <https://developer.android.com/guide/components/fundamentals.html>.

[9] AudioDeviceInfo. Android Developers [online]. 2018 [cit. 2018-05-10]. Dostupné z: <https://developer.android.com/reference/android/media/AudioDeviceInfo>.

[10] AudioFormat. *Android Developers* [online]. 2018 [cit. 2018-05-14]. Dostupné z: <https://developer.android.com/reference/android/media/AudioFormat>.

[11] AudioManager. *Android Developers* [online]. 2018 [cit. 2018-05-10]. Dostupné z: <https://developer.android.com/reference/android/media/AudioManager>.

[12] AudioRecord. *Android Developers* [online]. 2018 [cit. 2018-05-10]. Dostupné z: <https://developer.android.com/reference/android/media/AudioRecord>.

[13] AudioTrack. *Android Developers* [online]. 2018 [cit. 2018-05-10]. Dostupné z: <https://developer.android.com/reference/android/media/AudioTrack.html>.

[14] Better performance through threading. *Android Developers* [online]. 2018 [cit. 2018-04-30]. Dostupné z: <https://developer.android.com/topic/performance/threads>.

[15] Bitmap. *Android Developers* [online]. 2018 [cit. 2018-05-05]. Dostupné z: <https://developer.android.com/reference/android/graphics/Bitmap>.

[16] CameraCaptureSessionStateCallback. *Android Developers* [online]. 2018 [cit. 2018-05-01]. Dostupné z: <developer.android.com/reference/android/hardware/camera2/CameraCaptureSession.StateCallback>.

[17] CameraDevice. *Android Developers* [online]. 2018 [cit. 2018-05-01]. Dostupné z: <developer.android.com/reference/android/hardware/camera2/CameraDevice>.

[18] CameraDeviceStateCallback. *Android Developers* [online]. 2018 [cit. 2018-05-01]. Dostupné z: <developer.android.com/reference/android/hardware/camera2/CameraDevice.StateCallback>.

[19] CameraCharacteristics. *Android Developers* [online]. 2018 [cit. 2018-05-01]. Dostupné z: <developer.android.com/reference/android/hardware/camera2/CameraCharacteristics>.

[20] CameraManager. *Android Developers* [online]. 2018 [cit. 2018-05-01]. Dostupné z: <developer.android.com/reference/android/hardware/camera2/CameraManager>.

[21] CaptureRequest. *Android Developers* [online]. 2018 [cit. 2018-05-03]. Dostupné z: <https://developer.android.com/reference/android/hardware/camera2/CaptureRequest>.

[22] CaptureRequest.Builder. *Android Developers* [online]. 2018 [cit. 2018-05-03]. Dostupné z: <https://developer.android.com/reference/android/hardware/camera2/CaptureRequest.Builder>.

[23] Color. *Android Developers* [online]. 2018 [cit. 2018-05-05]. Dostupné z: <https://developer.android.com/reference/android/graphics/Color>.

[24] Dashboards. *Android Develoers* [online]. 2017 [cit. 2017-11-22]. Dostupné z: <https://developer.android.com/about/dashboards/index.html>.

[25] Detecting camera features with Camera2. Google developers - Medium [online]. 2016 [cit. 2018-05-01]. Dostupné z: <medium.com/google-developers/detecting-camera-features-with-camera2-61675bb7d1bf>.

[26] DIAS JUNIOR, A., S. MURALI, F. RINCON a D. ATIENZA. *Methods for reliable estimation of pulse transit time and blood pressure variations using smartphone sensors.* Microprocessors and Microsystems. 2016, 46, 84-95. ISSN 01419331.

[27] FABIÁN, Vratislav. *Neinvazivní měření krevního tlaku: Založené na oscilometrickém principu.* Praha, 2012. Disertační práce. České vysoké učení technické v Praze.

[28] FOO, Jong Yong Abdiel, Chu Sing LIM a Ping WANG. *Evaluation of blood pressure changes using vascular transit time. Physiological Measurement* [online]. 2006, 27(8), 685-694 [cit. 2017-12-28]. DOI: 10.1088/0967-3334/27/8/003. ISSN 0967-3334. Dostupné z: <http://stacks.iop.org/0967-3334/27/i=8/a=003?key=crossref.c2de29a3f7d7f06b3b3393f81756f7cd>.

[29] FONTANA, Josef et al. Funkce buněk a lidského těla [online]. Multimediální skripta. 2015 [cit. 2017-12-31]. <Dostupnéz:http://fblt.cz/>.

[30] GANONG, William F. *Přehled lékařské fysiologie.* Jinočany: H & H, 1995. ISBN 80-85787-36-9.

[31] GESCHE, Heiko, Detlef GROSSKURTH, Gert KÜCHLER a Andreas PATZAK. Continuous blood pressure measurement by using the pulse transit time: comparison to a cuff-based method. *European Journal of Applied Physiology* [online]. 2012, 112(1), 309-315 [cit. 2017-12-28]. DOI: 10.1007/s00421-011-1983-3. ISSN 1439-6319. Dostupné z: <http://link.springer.com/10.1007/s00421-011-1983-3>.

[32] How to get and set pixel value in Java. DYclassroom [online]. c2014-2018 [cit. 2018-05-05]. Dostupné z: <https://www.dyclassroom.com/image-processing-project/how-to-get-and-set-pixel-value-in-java>.

[33] CHANDRASEKARAN, V., R. DANTU, S. JONNADA, S. THIYAGARAJA a K. P. SUBBU. *Cuffless Differential Blood Pressure Estimation Using Smart Phones.* IEEE Transactions on Biomedical Engineering. 2013, 60(4), 1080-1089. ISSN 0018-9294.

[34] CHMELAŘ, Milan. *Lékařská přístrojová technika I.* Brno: Akademické nakladatelství CERM, 1995. Učební texty vysokých škol. ISBN 80-85867-63-X.

[35] Chroma Subsampling. *Rtings* [online]. 2017 [cit. 2018-05-06]. Dostupné z: <https://www.rtings.com/tv/learn/chroma-subsampling>.

[36] ImageFormat. *Android Developers* [online]. 2018 [cit. 2018-05-05]. Dostupné z: <https://developer.android.com/reference/android/graphics/ImageFormat>.

[37] ImageReader. *Android Developers* [online]. 2018 [cit. 2018-05-01]. Dostupné z: <developer.android.com/reference/android/media/ImageReader>.

[38] iPhone Microphone Frequency Response Comparison. *Faber Acoustical* [online]. 2009 [cit. 2018-01-02]. Dostupné z: <https://blog.faberacoustical.com/2009/ios/iphone/iphone-microphone-frequency-response-comparison>.

[39] Jak šel čas s operačním systémem android. *Android Tip* [online]. 2015 [cit. 2017-11-22]. Dostupné z: <http://www.androidtip.cz/jak-sel-cas-s-operacnim-systemem-android/>.

[40] JUNIOR, Alair Dias, Srinivasan MURALI, Francisco RINCON a David ATIENZA. Estimation of Blood Pressure and Pulse Transit Time Using Your Smartphone. In: *2015 Euromicro Conference on Digital System Design* [online]. IEEE, 2015, 2015, s. 173-180 [cit. 2018-05-18]. DOI: 10.1109/DSD.2015.90. ISBN 978-1-4673-8035-5. Dostupné z: <http://ieeexplore.ieee.org/document/7302267/>.

[41] KOLÁŘ, Radim. *Lékařská diagnostická technika.* Brno, 2006. Elektronická skripta. VUT Brno.

[42] Krevní tlak dle věku. *Tlakoměry* [online]. [cit. 2017-12-23]. Dostupné z: <https://www.tlakomery.cz/krevni-tlak-dle-veku-jake-jsou-optimalni-hodnoty>.

[43] LACKO, Luboslav. *Vývoj aplikací pro Android.* Brno: Computer Press, 2015. ISBN 978-80-251-4347-6.

[44] MediaRecorder. *Android Developers* [online]. 2018 [cit. 2018-05-01]. Dostupné z: <developer.android.com/reference/android/media/MediaRecorder>.

[45] Meet Android Studio. *Android Developers* [online]. 2017 [cit. 2017-11-27]. Dostupné z: <https://developer.android.com/studio/intro/index.html>.

[46] NĚMCOVÁ, Helena. *Měření krevního tlaku.* Medicína pro praxi [online]. 2007, (1), 7-12 [cit. 2017-12-29]. <Dostupnéz:https://www.medicinapropraxi.cz/pdfs/med/2007/01/02.pdf>.

[47] Pulse-code modulation. *In: Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 2018 [cit. 2018-05-15]. Dostupné z: <https://en.wikipedia.org/wiki/Pulse-code_modulation>.

[48] Renderscript Computation. *Android Developers* [online]. 2016 [cit. 2018-05-11]. Dostupné z: <https://stuff.mit.edu/afs/sipb/project/android/docs/guide/topics/renderscript/compute.html>.

[49] RenderScript Overview. *Android Developers* [online]. 2018 [cit. 2018-05-01]. Dostupné z: <developer.android.com/guide/topics/renderscript/compute>.

[50] ROKYTA, Richard. *Fyziologie pro bakalářská studia v medicíně, přírodovědných a tělovýchovných oborech.* Praha: ISV, 2000. Lékařství. ISBN 80-858-6645-5.

[51] Samsung Galaxy S7. *GSMArena* [online]. c2000-2018 [cit. 2018-05-19]. Dostupné z: <https://www.gsmarena.com/samsung_galaxy_s7-7821.php>.

[52] Surface. *Android Developers* [online]. 2018 [cit. 2018-05-01]. Dostupné z: <developer.android.com/reference/android/view/Surface>.

[53] System. *Android Developers* [online]. 2018 [cit. 2018-05-15]. Dostupné z: <https://developer.android.com/reference/java/lang/System>.

[54] SystemClock. *Android Developers* [online]. 2018 [cit. 2018-05-15]. Dostupné z: <https://developer.android.com/reference/android/os/SystemClock.html>.

[55] The Activity Lifecycle. *Android Developers* [online]. 2017 [cit. 2017-11-27]. Dostupné z: <https://developer.android.com/guide/components/activities/activity-lifecycle.html>.

[56] TROJAN, Stanislav. *Lékařská fyziologie.* Vyd. 4., přeprac. a dopl. Praha: Grada, 2003. ISBN 80-247-0512-5.

Seznam symbolů, veličin a zkratek

ACD Analogově digitální převodník - Analog-Digital Converter
API Aplikační programovací rozhraní - Application Programming Interface
CDD Dokument definice kompatibility - Compatibility Definition Document
CPU Centrální procesorová jednotka - Central Processing Unit
DPS Digitální signální procesor - Digital Signal Processor
GPU Grafický procesor - Graphic Processing Unit
GUI Grafické uživatelské rozhraní - Graphical User Interface
HW Technické vybavení - Hardware
IDE Integrované vývojové prostředí - Integrated Development Environment
NDK Nástroje pro nativní vývoj v jazyce *C* či *C++* - Native Development Kit
OOP Objektově orientované programování - Object-Oriented Programming
PCG Fonokardiograf - Phonocardiogram
PCM Pulzně kódová modulace - Pulse Code Modulation
PPG Fotopletysmograf - Photopletysmograph
PTT Čas přenosu pulzu - Pulse Transit Time
SDK Nástroje pro vývoj software - Software Development Kit
SW Programové vybavení - Software
TK Tlak krve, krevní tlak

Obsah přiloženého CD

Na přiloženém CD lze nalézt kódy vyvíjené Android aplikace, skript na výpočet hodnoty krevního tlaku v Matlabu a naměřená data.

- /
  - `Android aplikace` ..... nejdůležitější zdrojové kódy aplikace
    - `java`
    - `res`
    - `rs`
    - `bpdataacquisition.apk` ..... instalační soubor
  - `Android projekt` ..... projekt vyexportovaný z Android Studia
  - `Matlab`
    - `Data` ..... naměřená data
      - `Nesynchronizováno`
      - `User01`
      - `User02`
      - `User03`
      - `User04`
    - `BloodPressureEstimation.m` ..... skript pro výpočet krevního tlaku
  - `android-7.0-cdd.pdf` ..... dokument kompatibility pro verzi 7.0
  - `DP-Vařečka.pdf` ..... tato práce