VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ

Fakulta elektrotechniky
a komunikačních technologií

BAKALÁŘSKÁ PRÁCE

Brno, 2021 Jakub Matuška

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ
FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

ÚSTAV TELEKOMUNIKACÍ
DEPARTMENT OF TELECOMMUNICATIONS

BEZDRÁTOVÉ OVLÁDÁNÍ ROBOTA POMOCÍ MOBILNÍ PLATFORMY
WIRELESS ROBOT CONTROL USING MOBILE PLATFORM

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE
AUTHOR
Jakub Matuška

VEDOUCÍ PRÁCE
SUPERVISOR
Ing. Jiří Přinosil, Ph.D.

BRNO 2021

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

Bakalářská práce

bakalářský studijní program **Informační bezpečnost**

Ústav telekomunikací

Student: Jakub Matuška ***ID:*** 211804
Ročník: 3 ***Akademický rok:*** 2020/21

NÁZEV TÉMATU:

Bezdrátové ovládání robota pomocí mobilní platformy

POKYNY PRO VYPRACOVÁNÍ:

V rámci práce navrhněte a realizujte aplikaci pro ovládání robota vybaveného kamerou pro mobilní platformu s operačním systémem Android. Vlastní ovládání bude spočívat ve směrovém ovládání pohybu, vyčítání stavových informací a zobrazení obrazového signálu z kamery. Součástí práce bude rovněž realizace modulu řízení na straně robota. Pro komunikaci mezi robotem a mobilním zařízením navrhněte vhodnou koncepci zohledňující bezpečnost spojení a ochranu přenášených dat .

DOPORUČENÁ LITERATURA:

[1] ROGERS, Rick, et al. Android application development: Programming with the Google SDK. O'Reilly Media, Inc., 2009.

[2] HOWSE, Joseph. Android application programming with OpenCV 3. Packt Publishing Ltd, 2015.

Termín zadání: 1.2.2021 ***Termín odevzdání:*** 31.5.2021

Vedoucí práce: Ing. Jiří Přinosil, Ph.D.

doc. Ing. Jan Hajný, Ph.D.
předseda rady studijního programu

ABSTRAKT

Tato bakalářská práce se zabývá návrhem a implementací aplikace pro všesměrové ovládání pohybu robota pomocí mobilního zařízení s operačním systémem Android. Aplikace umožňuje i zobrazení obrazového signálu z webkamery robota. V rámci realizace byla vytvořena Android aplikace sloužící jako uživatelské rozhraní. Robota lze ovládat pomocí dvou virtuálních joystiků. Realizován byl i modul řízení na straně robota. Jako řídící jednotka byl použit mikropočítač Raspberry Pi. Služba streamování videa byla realizována pomocí multimediálního frameworku GStreamer v podobě RTP streamu. Byl realizován modul zabezpečení. V práci je předložen potřebný teoretický základ a následně je popsána samotná realizace aplikace.

KLÍČOVÁ SLOVA

Robot, Všesměrový pohyb, Mecanum kola, Raspberry Pi, Android, Streamování videa, Webkamera, GStreamer, Kryptografie, Digitální podpis, Java, Python, C++

ABSTRACT

This bachelor's thesis deals with design and implementation of an application for robot's omnidirectional movement control using mobile platform. Implementation includes an Android application, used as user interface, as well as robot–side Python program for controlling movement and sending RTP stream to Android application. User can control robot's movement using two virtual joysticks. Raspberry Pi was used as the control unit. The application has security module. Pipeline-based multimedia framework named GStreamer was used to implement RTP steaming. This paper describes necessary theory first and then introduces basic building blocks used in creation process of the application.

KEYWORDS

Robot, Omnidirectional movement, Mecanum wheels, Raspberry Pi, Android, Streaming, Webcam, GStreamer, Cryptography, Digital signature, Java, Python, C++,

Vysázeno pomocí balíčku `thesis` verze 4.03; `http://latex.feec.vutbr.cz`

MATUŠKA, Jakub. *Bezdrátové ovládání robota pomocí mobilní platformy*. Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav telekomunikací, 2021, 56 s. Bakalářská práce. Vedoucí práce: Ing. Jiří Přinosil, Ph.D.

Prohlášení autora o původnosti díla

Jméno a příjmení autora:	Jakub Matuška
VUT ID autora:	211804
Typ práce:	Bakalářská práce
Akademický rok:	2020/21
Téma závěrečné práce:	Bezdrátové ovládání robota pomocí mobilní platformy

Prohlašuji, že svou závěrečnou práci jsem vypracoval samostatně pod vedením vedoucí/ho závěrečné 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 závěrečné práce dále prohlašuji, že v souvislosti s vytvořením této závěrečné 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*

*Autor podepisuje pouze v tištěné verzi.

PODĚKOVÁNÍ

Rád bych poděkoval vedoucímu bakalářské práce panu Ing. Jířímu Přinosilovi, Ph.D. za odborné vedení, konzultace a podnětné návrhy k práci. Děkuji také své rodině a blízkým přátelům za podporu a pochopení.

Obsah

Seznam obrázků

Úvod

V rámci diplomové práce Štěpána Dvořáčka *Robot s autonomním audio-vizuálním řízením* [1] byl sestrojen robot s cílem autonomního řízení, tedy jeho řídící jednotka sama řídila pohyby robota podle vstupu z kamery a hlasových povelů. Tato práce si však klade za cíl přesunout řízení ze strany robota, na běžně prodávaný chytrý mobilní telefon a zajistit službu streamování videa z webkamery robota. Tak, aby mohl robota pohodlně ovládat kdokoliv, kdo je vlastníkem takového mobilního zařízení.

V současné době se všechny služby vytváří s ohledem na běžně užívané chytré telefony. Neboť jsou součástí našich každodenních životů a pomáhají nám v nemalých věcech. Spolu s jejich mobilitou je dnes většina lidí nosí neustále při sobě a tedy se staly jejich nepostradatelným nástrojem, pomáhajícím jim jak k užitku tak k zábavě. Využití chytrých telefonů začíná u běžného vyřizování emailů, okamžité komunikace kdekoliv, kdykoliv a s kýmkoliv, až po sledování aktivity kolem vašeho domu pomocí dohledových video systémů a ovládání vaší chytré domácnosti. Například i ovládání dronů a jiných RC modelů, dnes míří od klasických rádio vysílačů na mobilní platformy. Je tomu právě pro pohodlnost, výkonnost a všudypřítomnost chytrých mobilních zařízení. Toto je jeden z důvodů, proč řízení robota v této práci bude realizováno právě prostřednictvím mobilní platformy.

S rostoucí aktivitou, na této platformě, však roste i počet bezpečnostních incidentů – pokusů o nabourání se do dané aplikace či kompromitace jejich data. Je tedy potřeba dbát i na bezpečnost naší aplikace. Jedním z požadavků na řídící aplikaci tedy je, aby nejen umožňovala ovládání robota a přenos videa z kamery robota, ale také zajistit bezpečnost těchto služeb.

1 Popis robota

V této kapitole bude popsán vznik a původní účel robota, jeho konstrukce a změny provedené pro použití v této práci.

1.1 Původ a účel robota

Robot byl sestrojen v rámci diplomové práce Štěpána Dvořáčka Robot s *autonomním audio-vizuálním řízením* [1], která si kladla za cíl návrh a realizaci plně autonomního robota. Robot měl videokameru, díky které byl schopen za pomocí počítačového vidění a umělé inteligence rozeznávat obličeje osob a řídit robota tak, aby udržoval detekovaný obličej uprostřed snímaného obrazu. Dalším vstupem pro robota, byly hlasové příkazy. Robot měl mikrofon pro snímání těchto příkazů. Umělá inteligence se snažila rozpoznávat jednotlivé příkazy z řečového signálu. Takovými příkazy pak byly například „stop“, „dopředu“, „dozadu“, „doleva“ či „doprava“.

1.2 Popis originálního robota

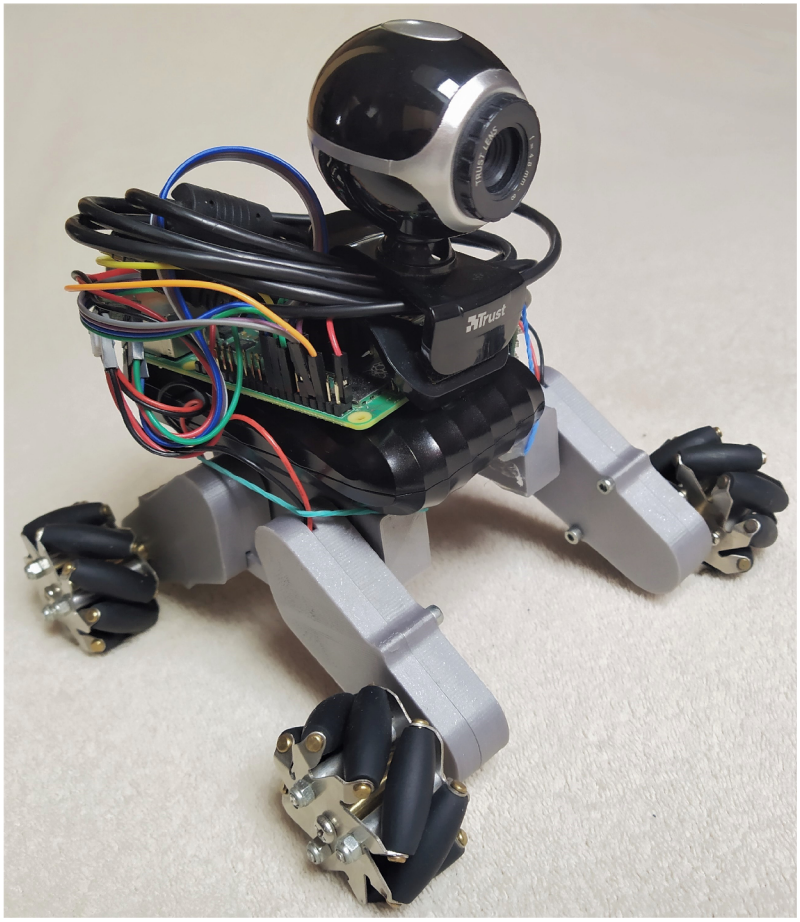
Kostra robota byla vytištěna na 3D tiskárně. Jako kola byla použita všesměrová Mecanum kola, která dávala robotovi specifickou charakteristiku pohybu. Každé kolo bylo poháněné vlastním DC[1] motorem s redukční převodovkou Pololu 1124 [2]. Jako řídící jednotka byl využit mikropočítač ODROID U2, který pomocí sériové linky UART komunikoval s mikrokontrolérem STM32F042, jehož úkolem bylo vytvářet PWM[2] signál, pro regulaci intenzity motorů. Mikrokontrolér STM32F042 generoval společně s PWM signálem i signál určující směr otáčení kola. Tyto signály byly přivedeny na motor kontrolér DRV8835 [3], který pomocí H-můstků dokázal plynule regulovat otáčky DC motorů. Jeden motor kontrolér dokázal regulovat 2 motory, proto byly na desku robota osazeny dva, pro celkové ovládání 4 DC motorů. Napájení celého robota, bylo zajištěno dvěma Lipol bateriemi typu 18650 zapojenými do série s výslednými parametry 7,2V/2400mAh. Podrobnější informace mohou být nalezeny v původní práci [1].

1.3 Změny pro použití v této práci

Výše popsaný robot byl převzat pro demonstraci dálkového ovládání robota pomocí mobilní platformy v této práci. Na robotovi byly provedeny úpravy a to za účelem

[1] Direct current – stejnosměrný proud
[2] Pulse-width modulation – pulzně šířková modulace

lepší kompatibility, spolehlivosti a pohodlnosti obsluhy robota. Hlavními změnami jsou, výměna řídícího mikropočítače ODROID U2 za Raspberry Pi 4 Model B [4] a výměna původních Lipol baterii za powerbanku s dvěma proudově oddělenými výstupy s parametry 5V/10050mAh a maximálním proudem 2.4A. Jeden z výstupů powerbanky je použit pro napájení Raspberry Pi a druhý pro napájení 2 motor kontrolérů. Pokles napětí a proudu má za příčinu slabší výkon DC motorů a tedy robot nedosahuje plné rychlosti jako původní. Avšak robot má stále velmi pěkné výsledky, plně postačující pro účely demonstrace ovládání robota v této práci. Robot na rovném povrchu a plném výkonu všech motorů dosahuje rychlosti 0,25 m/s.

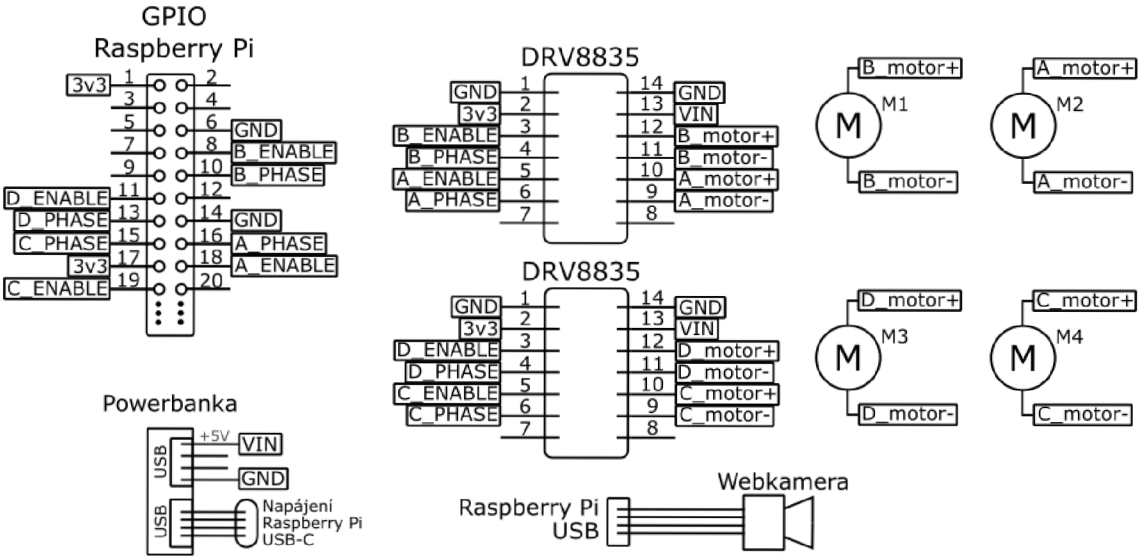


Obr. 1.1: Robot

Nový mikropočítač Raspbbery Pi již zvládá generovat PWM signál i signál určující směr otáčení sám a již tedy není nutný řídící mikrokontrolér STM32F042. Byl tedy z obvodu vyřazen a PWM signál je veden z GPIO (General-purpose input/output) pinů na desce Raspbbery Pi rovnou na 2 motor kontroléry, zajišťující plynulé řízení otáček 4 DC motorů.

Po těchto úpravách již nebyl potřeba původní plošný spoj, zajišťující konektivitu jednotlivých prvků a převod napájecího napětí. Tento plošný spoj byl tedy z ro-

bota odebrán. Robotu byla přidána webkamera, připojená USB (Universal Serial Bus) konektorem. Poslední úpravou bylo odebrání funkcionality autonomního řízení a její nahrazení aplikací vytvořenou a dále popsanou v této práce. Aktuální zapojení robota je vidět na obrázku 1.2. Samotný robot je pak vyfocen na 1.1



Obr. 1.2: Zapojení robota

2 Zabezpečení

Tato kapitola pojednává o možnostech zabezpečení aplikace. Rozebírá možné hrozby, jejich dopady při zneužití a možnosti jejich ochrany. Zároveň předkládá teoretický základ pro realizaci těchto ochran.

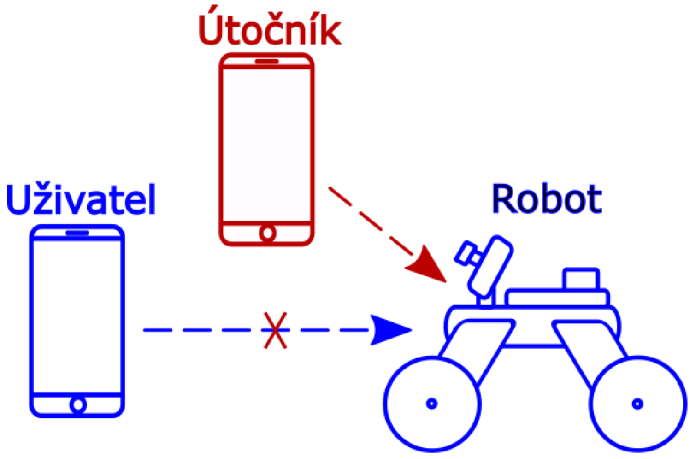
2.1 Možné hrozby útoků

Zde je hrubý přehled možných útoků a jejich dopady na naši aplikaci. U každé hrozby je i nastíněný způsob, jak danou hrozbu minimalizovat.

2.1.1 Neautorizované ovládání robota

Hlavní hrozbou je neautorizované, neoprávněné ovládání robota. Tato situace může nastat v případě, kdy řídící jednotka nedokáže rozeznávat původ příchozích zpráv a tedy nedokáže ani rozeznat útočníka do legitimního uživatele. Robota by tak mohl ovládat kdokoliv. Při použití bezdrátové komunikace může kdokoliv přistupovat ke komunikačnímu médiu a tedy se snažit ovládat robota nebo zamezit našim zprávám úspěšné doručení. Pokud by se útočníkovi podařilo blokovat naše zprávy tak, aby se k robotu nedostaly, mohl by robota on sám plně ovládat. Ovládnutí robota útočníkem znázorňuje obrázek 2.1.

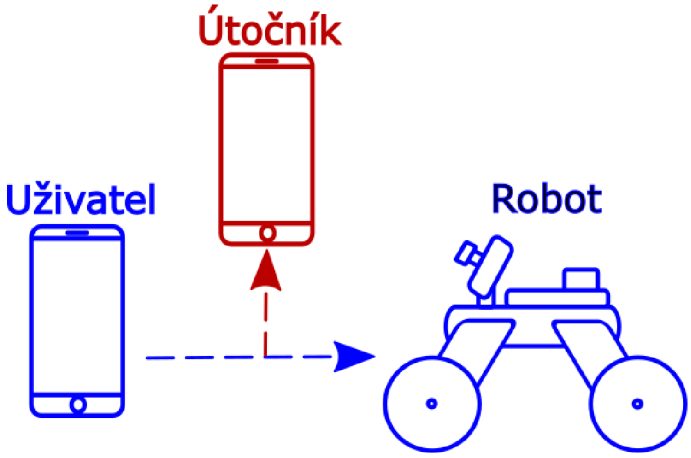
Z tohoto důvodu je potřeba autentizace zdroje přijatých zpráv tak, aby mohla řídící jednotka rozlišovat zdroje přijatých zpráv a přiřazovat jim jejich odpovídající práva. Tedy na základě výsledku autentizace provádět autorizaci a případnému útočníkovi tak odepřít práva k ovládání robota.



Obr. 2.1: Neautorizované ovládání robota

2.1.2 Odposlech komunikace

Další hrozbou je odposlech komunikace mezi klientem a robotem. Odposlechem by bylo možné nahlížet do zasílaných zpráv a dozvědět se jejich obsah. V našem případě by to znamenalo, že by útočník přesně věděl, kdy a jaké příkazy jsou zasílány řídící jednotce a v případě streamování videa by si mohl zkonstruovat celý stream a tedy sledovat přímý přenos videa z robota. Tím, že probíhající komunikace je bezdrátová, volně přístupná všem v dosahu signálu, by odposlech mohl provádět kdokoliv v tomto dosahu. Zabránit odposlechu můžeme šifrováním zasílaných zpráv. Útočník by tak sice stále mohl odposlouchávat, ale zprávy které by získal, by musel nejprve dešifrovat. Toho by neměl být schopen. Logické schéma odposlechu komunikace je vidět na obrázku 2.2.



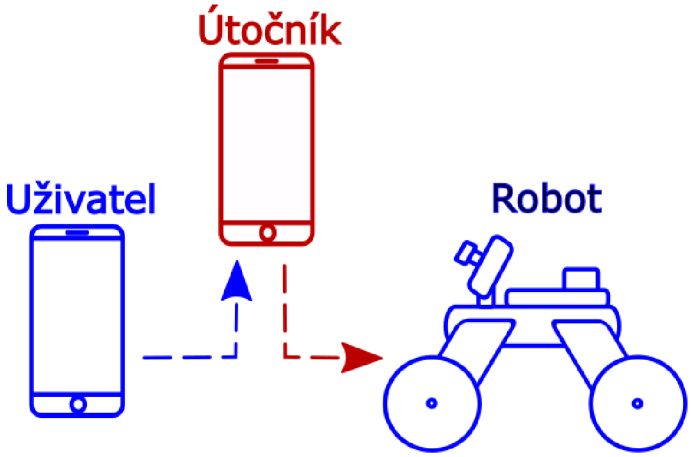
Obr. 2.2: Odposlech komunikace

2.1.3 Modifikace zasílaných zpráv

Případný útočník by mohl odchytávat zprávy, zasílané řídící jednotce a naopak. Ty by pak mohl pozměnit a vydávat je za původní. Tím by mohl například měnit směr jízdy robota. Tomuto útoku se říká útok mužem uprostřed (Man in the Middle, MitM). Tímto útokem by byla porušena integrita komunikace, to lze rozpoznat a buď změněné zprávy zahazovat nebo se je snažit opravit. Integritu lze zajistit například algoritmy zakládajícími se na digitálních podpisech nebo kontrolních součtech, takzvaných hash kódech.

2.2 Hash funkce

Hashovací funkce je jednocestná, bezkolizní funkce $h()$, která vstupní řetězec M libovolné délky převede na výstupní řetězec pevné délky podle typu hashovací funkce.



Obr. 2.3: Modifikace komunikace

Cílem hashovací funkce je převést vstupní řetězec na výstupní tak, aby se z výstupního řetězce nedal odvodit řetězec vstupní (jednocestnost). Dále by mělo být jednoduché vypočítat *h(M)*, ale velmi obtížné by měl být výpočet inverzi této funkce, tedy z výstupu *h(M)* určit původní zprávu *M*. Pro stejný vstup by funkce měla dát vždy stejný výstup.

Hashovací funkce by měla být také bezkolizní, neměli by se tedy dva různé vstupy *M* a *M'* mapovat na stejný výstup. Vstupní množina je však nekonečná a výstupní množina je omezena délkou výstupu, proto kolize vždy nastane. Měla by tedy být zajištěna pouze složitost nalezení takové kolize. Nejpoužívanější hash funkce a délky jejich výstupů jsou MD5 (128b), SHA-1 (160b), SHA-2 (224, 256, 384, 512b), SHA-3 (výstupy jako u SHA-2). Podrobně se hash funkcemi zabývá [5].

2.3 Autentizace

Autentizace je proces ověřování, zda je daná entita opravdu entitou za kterou se vydává. Jinak řečeno autentizace je ověření identity a sice identity uživatele, procesu (služby) nebo stroje (serveru, komunikační jednotky). Autentizujeme abychom měli jistotu s kým komunikujeme. Autentizace konkrétního uživatele (identity) probíhá pomocí autentizační metody. Ta využívá autentizační faktory jako je například vědomost tajné informace nebo vlastnění nějakého předmětu. Spojením více autentizačních metod vzniká vícefaktorová autentizace například dvoufaktorová nebo třífaktorová. Příkladem dvoufaktorové autentizace je ověření identity (člověka) v internetovém bankovnictví pomocí znalosti tajné informace (hesla) a vlastněného předmětu (telefonu, ověřeného pomocí SMS[1] kódu).

[1]Short Message Service

2.3.1 Autentizace znalostí

Identita je ověřena na základě znalosti, kterou by identita měla vědět.

Nejvíce rozšířenou variantou je znalost tajného hesla, popřípadě kombinace přihlašovacího jména a hesla. Identita tyto údaje zasílá spolu s žádostí o přístup k aktivům. Autentizační entita (např. server) porovná přijaté heslo (nebo jeho hash) s heslem uloženým v databázi a buď přístup povolí nebo odmítne. Slabina této varianty spočívá ve složitosti hesla, tedy jak je heslo těžké prolomit (uhádnout). Hesla by měla být dostatečně složitá na to, aby průměrná doba prolomení hesla měla větší hodnotu než hodnota získaná prolomením hesla, aby se útok nevyplatil. Další slabinou může být zasílání hesla přes veřejnou síť, kde by mohlo být odchyceno a zneužito.

Tuto problematiku řeší varianta výzvy a odpovědi. Kde znalostí je opět kombinace uživatelského jména a hesla, nyní se však spolu s žádostí o přístup k aktivům zašle pouze uživatelské jména. Autentizační entita pak zašle zpět ověřované identitě výzvu, identita vytvoří odpověď s pomocí svého hesla a přijatou výzvou. Odpověď odešle zpět autentizační entitě a ta ověří správnost odpovědi. Tímto není heslo posíláno přes veřejnou síť, zůstává pouze na koncových stanicích. Další variantou ověření je důkaz nulové znalosti. V této variantě jde o prokázání znalosti bez přímého odhalení znalosti. Autentizační entita se dotazuje ověřované identity na sérii otázek, ze kterých určí s jakou pravděpodobností daná identita opravdu znalostí disponuje či nikoliv. Výhodou je vyloučení odcizení znalosti v průběhu auntentizace, nevýhodou však je nutnost více zaslaných zpráv a pouze pravděpodobnostní výsledek autentizace.

2.3.2 Autentizace předmětem

Identita se ověřuje na základě předmětu, jež vlastní. Tedy autentizačním faktorem je předmět například čipová karta, USB token nebo mobilní telefon. Identita pak předloží jedinečný předmět autentizační entitě. Entita zkontroluje originalitu předmětu a autentizace je hotova. Velkou nevýhodou je možnost ztráty/zcizení předmětu. Proto se často používá v kombinaci s autentizací znalostí v podobě dvoufaktorové autentizace.

2.3.3 Autentizace biometrikou

Identita se ověřuje tím čím je. U tohoto typu autentizace je ověřovanou entitou výhradně člověk. Ten má jedinečné rysy odlišující ho od ostatních lidí. Takovými fyziologickými rysy mohou být například otisky prstů, oční duhovka, oční sítnice, obličej a DNA. Behaviorálními rysy pak hlas, rukopis, chůze apod. Jednotlivé rysy

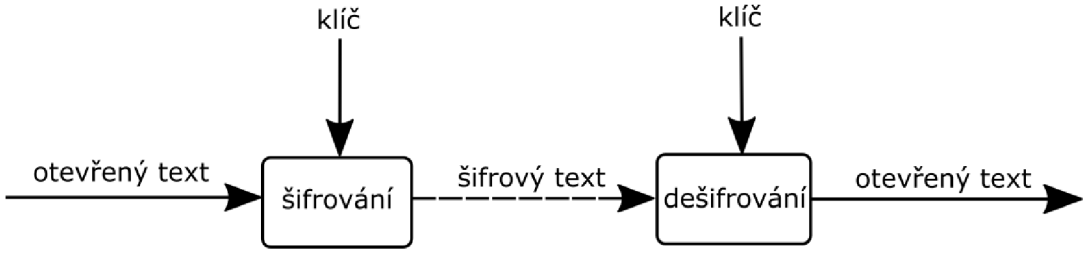
mají specifické způsoby jejich snímání. Autentizace pak spočívá v nasnímání daného rysu a jeho porovnání se záznamem v databázi.

2.4 Šifrování komunikace

Šifrování zpráv zasílaných mezi koncovými stanicemi slouží pro zajištění důvěrnosti dat. Tedy zajistí, že ten kdo nedokáže dešifrovat šifrový text, nezjistí co bylo původně zašifrováno. Šifrový text by neměl prozradit nic o původním textu a tedy šifrový text by se měl jevit jako náhodně vygenerovaná posloupnost. Převod z otevřeného (původního/nezašifrovaného) textu na text šifrový zajišťuje šifrovací funkce. Naopak šifrový text zpět na otevřený (plain) text, převádí dešifrovací funkce. Existuje mnoho šifrovacích algoritmů a jejích hlavní členění je na symetrické a asymetrické [6].

2.4.1 Symetrické algoritmy

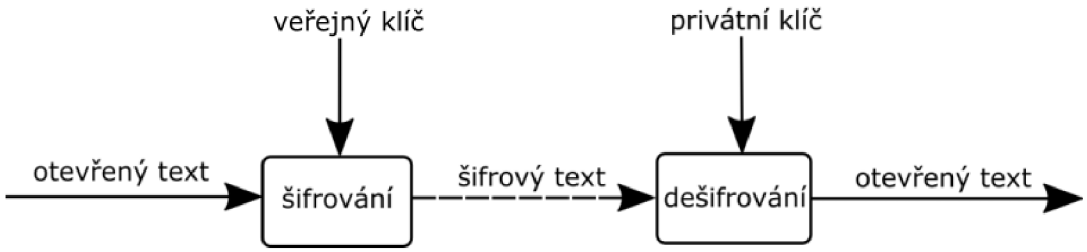
Symetrické algoritmy používají jeden klíč, jak pro šifrování tak dešifrování zprávy. Ten kdo daný klíč zná je schopen, jak šifrovat, tak i dešifrovat. Je tedy důležité, aby klíč znaly pouze oprávněné entity. Čímž vzniká problém ukládání a distribuce klíče tak, aby se nedostala do rukou neoprávněným entitám. Tyto algoritmy jsou však velmi rychlé v porovnání s asymetrickými.



Obr. 2.4: Symetrické šifrování

2.4.2 Asymetrické algoritmy

Asymetrické algoritmy používají 2 klíče, jeden pro šifrování a druhý pro dešifrování. K šifrování se používá takzvaný veřejný klíč přístupný komukoliv a k dešifrování takzvaný privátní (utajovaný) klíč. Šifrovat tedy může kdokoliv a dešifrovat pouze vlastník privátního klíče. Privátní klíč nesmí být sdílen a měl by být bezpečně uchován na zařízení, kde byl vygenerován. Naopak veřejné klíče jsou distribuovány veřejnosti například za pomocí certifikátů.

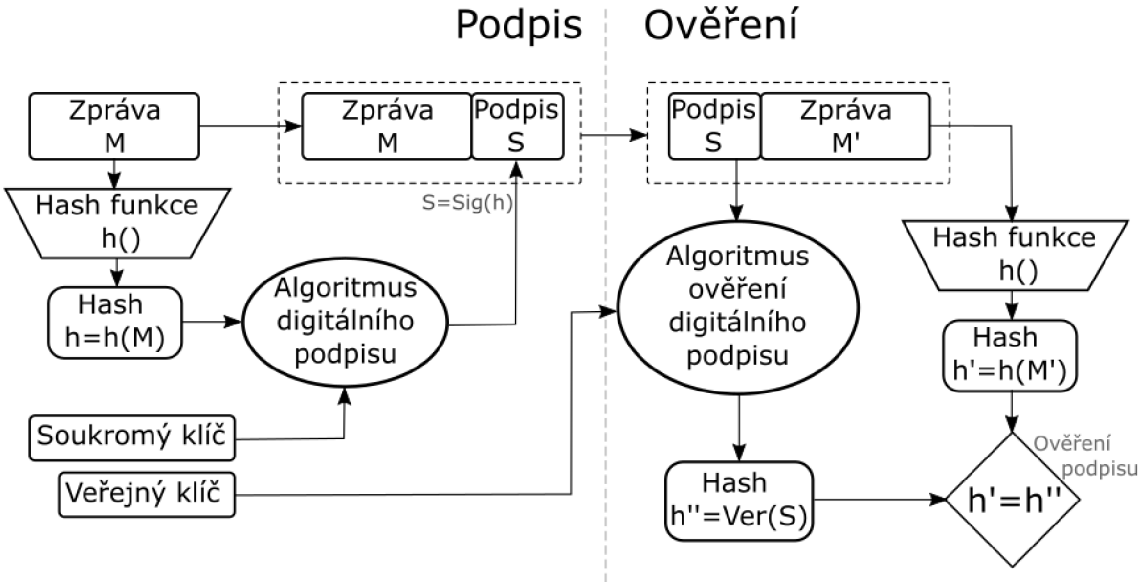


Obr. 2.5: Asymetrické šifrování

2.5 Digitální podpis

Digitální podpis by měl mít stejné vlastnosti a sloužit pro stejný účel jako podpis klasický (ruční). Měl by tedy být nefalšovatelný, nepřenositelný, nepopiratelný, podepsaný dokument by nemělo být možné měnit a měl by sloužit k autentizaci podepisující identity.

Obecný postup digitálního podpisu zprávy M je následující. Ze zprávy M se vytvoří hash $h = h(M)$, ten se podepíše zvoleným algoritmem a vznikne digitální podpis S. Tím je fáze podpisu hotová v podobě zprávy M a jejího podpisu S. Při ověřování se pak vezme podpis S, ověří se podle odpovídajícího algoritmu, který byl zvolen při vytváření podpisu S a tím dostaneme hash h'. Pokud $h' = h = h(M)$, tedy pokud se získaný hash h' rovná hashy z přijaté zprávy M, podpis je správně ověřen a my víme, že zpráva je od podepisující indenty a nebyla změněna. Logické schéma podpisu a ověření je na obrázku 2.6. Nejpoužívanějšími algoritmy pro digitální podpis jsou RSA, DSA a ECDSA.



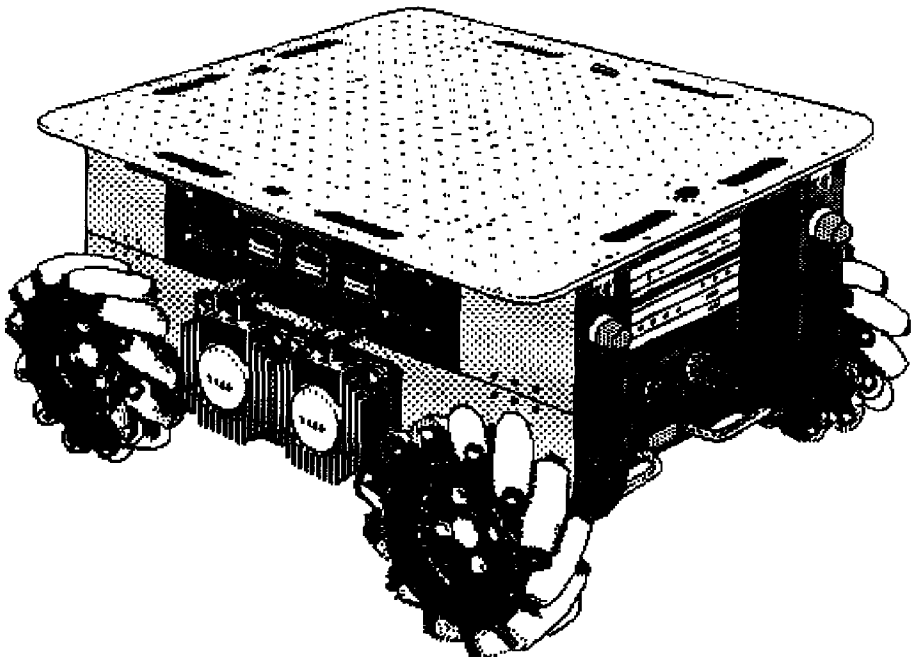
Obr. 2.6: Schéma digitálního podpisu

3 Možnosti pohybu

Tato kapitola popíše charakteristiku pohybu robota umožněnou všesměrovými koly Mecanum. Nastíní teorii jak Mecanum kola fungují a rozebere možnosti ovládání pohybu pomocí mobilního telefonu. Popíše jejich výhody a nevýhody a nakonec vybere jeden z navržených způsobů ovládání pohybu robota, tak aby co nejlépe vyhovoval naší aplikaci.

3.1 Všesměrová kola Mecanum

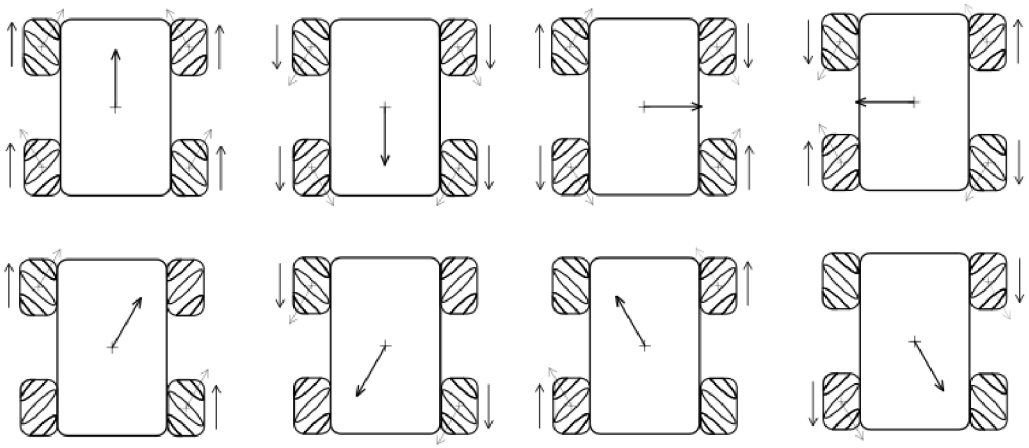
Všesměrová kola Mecanum byla vynalezena ve Švédsku a byl to Bengt Ilon, kdo si je roku 1972 patentoval [7]. Jedním z prvních zkonstruovaných robotů se všesměrovými koly Mecanum byl robot URANUS z univerzity Carnegie Mellon v roce 1990 [8]. Mecanum kolo se skládá z menších válečků zužujících se k oběma koncům. Válečky svírají úhel 45° s osou rotace kola a volně se protáčí. Kola jsou pevně orientována paralelně s podvozkem a mění pouze svou úhlovou rychlost. Při montáži na čtyřkolový podvozek je nutné dbát správné orientace kol a to tak, aby pomyslné osy malých válečků směřovaly do středu podvozku.



Obr. 3.1: Robot URANUS [8]

3.1.1 Kinematika Mecanum kol

Tím, že válečky svírají s osou otáčení kola 45° a volně se protáčí, tak výsledný vektor síly není rovně podél kola, jak je tomu u normálních kol. Ale je paralelní s normálou osy válečku, což je úhel 45° od osy otáčení celého kola a má velikost podle úhlové rychlosti kola. Součtem vektorů všech 4 kol získáme výsledný vektor pohybu celého robota. Při použití stejné rychlosti pro všechna kola, dokážeme pohybovat robotem do 8 směrů, bez nutnosti měnit orientaci robota. Orientace otáčení kol, vektory síly jednotlivých kol a jejich výsledné součty určující pohyb robota, pro oněch 8 směrů, jsou uvedeny na obrázku 3.2.



Obr. 3.2: Základních 8 směrů robota

Ostatní směry pohybu mohou být dosaženy individuálním nastavením rychlosti pro jednotlivá kola. Dané rychlosti mohou být vypočteny podle vztahu 3.1. [9][10]

Kde ω_i je úhlová rychlost kola i ($i = \{1, 2, 3, 4\}$); ω_z je úhlová rychlost robota kolem svislé osy (rotace); l_x a l_y jsou vzdálenosti středu kola od středu podvozku; r je poloměr kola a v_x a v_y jsou jednotlivé složky výsledného vektoru pohybu robota $v = (v_x, v_y)$.

$$\begin{bmatrix} \omega_1 \\ \omega_2 \\ \omega_3 \\ \omega_4 \end{bmatrix} = \frac{1}{r} \begin{bmatrix} 1 & -1 & -(l_x + l_y) \\ 1 & 1 & (l_x + l_y) \\ 1 & 1 & -(l_x + l_y) \\ 1 & -1 & (l_x + l_y) \end{bmatrix} \begin{bmatrix} v_x \\ v_y \\ \omega_z \end{bmatrix} \tag{3.1}$$

Převedeno na jednotlivé úhlové rychlosti pak:

$$\omega_1 = \frac{1}{r}(v_x - v_y - (l_x + l_y)\omega_z), \tag{3.2}$$

$$\omega_2 = \frac{1}{r}(v_x + v_y + (l_x + l_y)\omega_z), \tag{3.3}$$

$$\omega_3 = \frac{1}{r}(v_x + v_y - (l_x + l_y)\omega_z), \tag{3.4}$$

$$\omega_4 = \frac{1}{r}(v_x - v_y + (l_x + l_y)\omega_z), \tag{3.5}$$

Pokud víme naopak jednotlivé úhlové rychlosti kol, pak výsledný pohyb lze spočítat podle vztahu 3.6.

$$\begin{bmatrix} v_x \\ v_y \\ \omega_z \end{bmatrix} = \frac{r}{4} \begin{bmatrix} 1 & 1 & 1 & 1 \\ -1 & 1 & 1 & -1 \\ -\frac{1}{(l_x+l_y)} & \frac{1}{(l_x+l_y)} & -\frac{1}{(l_x+l_y)} & \frac{1}{(l_x+l_y)} \end{bmatrix} \begin{bmatrix} \omega_1 \\ \omega_2 \\ \omega_3 \\ \omega_4 \end{bmatrix} \tag{3.6}$$

Převedeno na jednotlivé složky pak:

$$v_x = (\omega_1 + \omega_2 + \omega_3 + \omega_4)\frac{r}{4} \tag{3.7}$$

$$v_y = (-\omega_1 + \omega_2 + \omega_3 - \omega_4)\frac{r}{4} \tag{3.8}$$

$$\omega_z = (\omega_1 + \omega_2 + \omega_3 + \omega_4)\frac{r}{4(l_x + l_y)} \tag{3.9}$$

3.1.2 Zhodnocení Mecanum kol

Hlavní výhodou těchto kol je pohyb do všech směrů bez nutnosti měnit orientaci robota. To zajišťuje robotovi snazší pohyb v úzkých prostorech. Používají se tak ve skladech na vysokozdvižných vozících nebo v nemocničním prostředí na lehátka nebo kolečková křesla [11]. Značnou nevýhodou je velká závislost na povrchu po kterém robot jede. Robot pro dobré vlastnosti potřebuje přiléhavý a co nejrovnější povrch. Není tedy vhodný do terénu. Další nevýhodou je neefektivita určená vyosenými vektory vůči směru pohybu. V našem případě se nám však kola hodí převážně kvůli schopnosti udržet zorné pole kamery jedním směrem za současného pohybu směrem jiným.

3.2 Ovládání pohybu robota

V této podkapitole budou popsány možnosti ovládání robota pomocí dotykového telefonu. Jelikož nejsme limitováni fyzickým charakterem pohybu, bude důležitý výběr způsobu ovládání robota, abychom této vlastnosti využili. Pohyb robota je řízen z Android aplikace a tedy budou ukázány jednotlivé možnosti řízení pohybu robota za současného zobrazování přímého přenosu pohledu z kamery robota na pozadí.

3.2.1 Směrové šipky

Směrové šipky jsou prvním a nejjednodušším řešením. Určitý počet tlačítek předává informaci o jim přiřazeném směru pohybu. Počet šipek je však omezen jejich velikostí na obrazovce, aby se příjemně mačkala a nezabírala příliš plochy, neboť na pozadí běží stream videa z webkamery. Touto variantou se tedy omezujeme pouze na předdefinované směry pohybu robota a předdefinovanou rychlost. Návrh aplikace s využitím směrových šipek je vidět na obrázku 3.3.

Obr. 3.3: Návrh směrových šipek

3.2.2 Joystick

Další možností je využití 2 virtuálních joysticků jako vstupů pro řízení pohybu robota. Jeden joystick pro všesměrový pohyb bez rotace a druhý pouze pro rotaci. Tak by byly pokryty všechny možnosti pohybu robota včetně možnosti určení intenzity. Toto řešení je značně složitější o proti směrovým šipkám, neboť by se museli vypočítávat jednotlivé úhlové rychlosti pro každý motor na základě pozic joysticků. Návrh aplikace s využitím 2 virtuálních joystiků je zobrazen na obrázku 3.4. Na pozadí bude přehráván stream z webkamery robota.

Obr. 3.4: Návrh joysticků

3.2.3 Řízení pomocí gest

Tato varianta je podobná virtuálním joystickům s tím rozdílem, že vstup není brán z joysticků, ale z gest vytvořených prstem na dotykové obrazovce. Pokud by tedy uživatel potáhl prstem kdekoliv na obrazovce kupředu, aplikace by toto gesto brala jako signál k pohybu robota vpřed. Takto by šla udělat gesta reprezentující všechny směry. Případně by šlo vypočítat pomocí počátku a konce gesta vektor chtěného pohybu robota a určit tak směr i intenzitu.

3.2.4 Naklánění telefonu

Tato varianta používá jako vstup gyroskopické senzory telefonu. Pro pohyb vpřed by stačilo telefon naklonit dopředu a totéž by platilo pro ostatní směry. Byl by však problém s určení některých směrů a naklánění obrazovky by mohlo vadit v pohledu na displej.

3.2.5 Výběr způsobu ovládání robota

V rámci této práce byl zvolen způsob ovládání pohybu robota za pomocí dvou virtuálních joysticků. Především pro možnost detailního určení směru i rychlosti pohybu robota. Pravý joystick slouží pro všesměrový pohyb po rovině bez rotace. A levý joystick naopak pouze pro rotaci. Aplikace je orientovaná naležato. Joysticky tak lze ovládat pomocí levého a pravého palce. Pocit z ovládání je tak podobný jako u klasického gamepadu na herních konzolích.

4 Vývoj Android aplikací

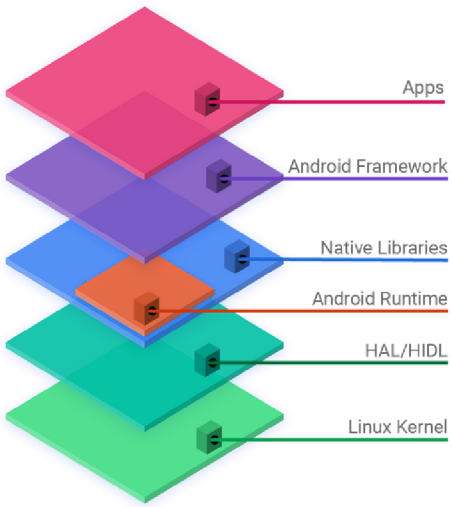
Tato kapitola pojednává o operačním systému Android, jeho struktuře, vlastnostech a samotném vývoji Android aplikací [12].

4.1 Operační systém Android

Operační systém (OS) Android byl vyvinut převážně firmou Google, je odvozen od Linuxového jádra a je primárně mířený na zařízení s dotykovou obrazovkou.

Pro nativní vývoj aplikací bylo vytvořeno vývojové prostředí Android Studio. Pro psaní Android aplikací se používají programovací jazyky jako jsou Kotlin, Java nebo C++. Android Studio využívá gradle pro udržování externích závislostí na jiné knihovny a moduly.

Celá architektura je na bázi stacku, kde nejnižší vrstvy propůjčují svojí funkcionalitu vyšším. Android software stack je vidět na obrázku 4.1. Skládá se od spodní vrstvy z Linuxového jádra, nativních knihoven a Android Runtime, Aplikačního frameworku a konečně samotných aplikací. Kód aplikace napsaný v Android studiu je nejprve zkompilován na takzvaný Java bytecode následně pak na Dalvik bytecode, který Dalvik zpracovává a interpretuje [13]. Dalvik je virtuální přístroj (virtual machine), podobný Java virtuálnímu přístroji (Java Virtual Machine), na zpracování a interpretování připraveného bytecodu. Nachází se na vrstvě Android Runtime v Android stacku.



Obr. 4.1: Android stack [14]

4.2 Hlavní koncept aplikací

Aplikace jsou zcela řízený OS Android a sice v podobě jejich životních cyklů. Je tomu převážně z důvodu, co největší úspory baterie, neboť u mobilních zařízení je baterie velmi omezený zdroj. Aplikace jsou tedy spuštěny pouze, pokud jsou potřeba a po splnění účelu jsou ihned vypnuty.

4.2.1 Oddělení jednotlivých aplikací

Každá aplikace žije ve vlastním takzvaném bezpečnostním sandboxu. Na každou aplikaci se OS dívá jako na jedinečného uživatele, kterému přiřadí jeho jedinečné uživatelské identifikační číslo (user ID) a na základě těchto ID jim přiděluje přístupová práva, defaultně to jsou jejich vlastní soubory. Každá aplikace má svůj vlastní virtual machine a tedy kód aplikace běží v absolutní izolaci od ostatních. Každá aplikace běží ve svém vlastním linuxovém procesu, OS spustí daný proces pouze pokud je potřeba interpretovat část kódu aplikace. Jakmile je interpretace hotová, proces je ihned zastaven. Zastaven může být i z důvodu potřeby výpočetních nebo paměťových prostředků používaných danou aplikací pro potřeby operačního systému. Defaultně je aplikaci přiděleno co nejméně práv. Jsou ji tedy přidělena práva pouze potřebná pro vykonání jejího účelu. O všechna práva musí aplikace dostat potvrzení od uživatele. Takto je vytvořeno bezpečné prostředí, kde je každá aplikace o samotě a dělá pouze to co má. Je ovšem žádoucí, aby aplikace využívaly již vytvořenou funkcionalitu jak OS, tak jiných aplikací. Mají pro to specifické prostředky jako je například Service komponenta.

4.2.2 Komponenty

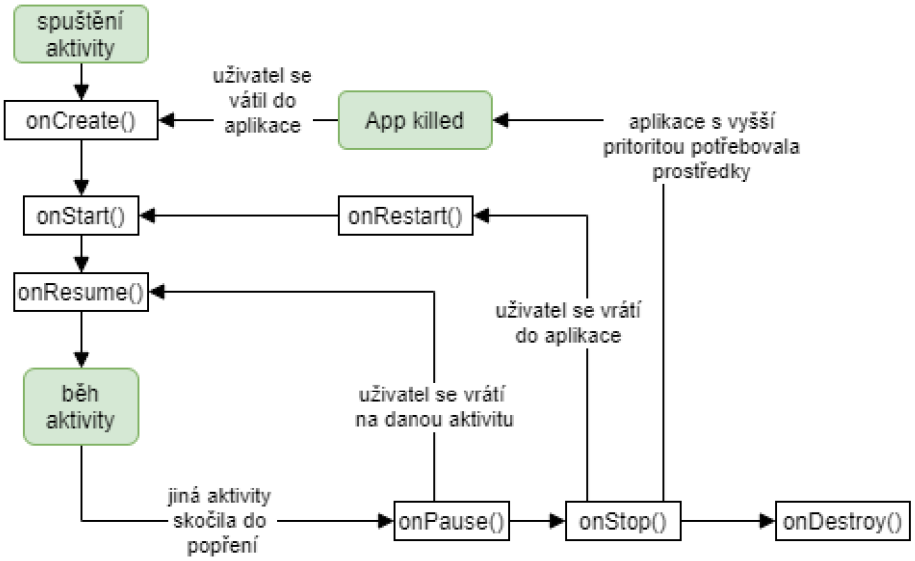
Aplikace se skládají z jednotlivých komponent. Takovými komponenty mohou být aktivity (Activities), služby (Services), Broadcast receivers a Content providers. Aktivita je komponenta sloužící jako uživatelské rozhraní, resp. složením více aktivit můžeme získat uživatelské rozhraní. Služba je komponenta zajišťující běh aplikace na pozadí, obsluhuje tedy funkcionality dané aplikace dostupné i v případě, že není zrovna využíváno hlavní uživatelské (UI[1]) vlákno. Tedy uživatel má aplikaci pouze v pozadí a vykonává se kód jiné aplikace. Toto je velmi dobře využitelné například pro realizaci sítového připojení dané aplikace. Funkcionality přístupné prostřednictvím služeb mohou být nabízeny i pro jiné aplikace. Broadcast receiver je komponenta umožňující zachytávání námi nebo operačním systémem vytvořené události

[1]User Interface thread – uživatelské vlákno je označení pro hlavní vlákno, které je na popředí a zobrauje jednotlivé aktivity uživateli

(eventy). Na tyto události pak může aplikace patřičně reagovat. Content provider slouží aplikaci pro přístup k datům.

4.2.3 Android Activity Lifecycle

Všechny aktivity podléhají jejich životnímu cyklu Android Activity Lifecycle [13]. Jednotlivé fáze se mění, jak uživatel prochází aplikací a jednotlivé aktivity se dostávají do popředí resp. do pozadí. Aktivity na jednotlivé změny mohou reagovat pomocí událostí. Zjednodušený životní cyklus aktivity i s jednotlivými názvy volaných událostí je popsán na obrázku 4.2.



Obr. 4.2: Android Activity Lifecycle

4.3 Android NDK

Android NDK (Nativ Development Kit) je soubor nástrojů umožňující použití jazyka C a C++ pro implementaci části aplikace. Současně přidává knihovny pro práci s fyzickými komponenty zařízení, jako jsou například senzory. Oficiální Android NDK dokumentace [15] říká, že se tento nástroj nehodí pro většinu aplikací neboť nadměrně zvyšuje komplexnost celé aplikace. Možné vhodné použití jsou v případě, kdy je potřeba velké výkonové optimalizace například pro hry nebo v případě využívání funkcionalit napsaných jinými vývojáři v podobě knihoven. Druhý důvod a sice použití již vytvořené knihovny pro přenos streamu, je důvod proč budeme i my používat Android NDK v naší aplikaci. Kód jazyku C a C++ je potřeba před použitím zkompilovat. K tomu nám poslouží vestavěný ndk-build a jeho makefiles

ve spolupráci s Gradlem. Jimi se vytvoří sdílené knihovny s koncovkou .so a pomocí Java kódu, pak můžeme volat, tyto funkce ve sdílených knihovnách pomocí JNI (Java Native Interface) frameworku, sloužícího jako rozhraní mezi Java a C++ kódem.

5 Bezdrátové sítě

Bezdrátové sítě tu s námi jsou od roku 1992. Od té doby se tato technologie velmi vyvinula a zrychlila. Vznikl standard IEEE 802.11 popisující konkrétní technologie [16].

Hlavní výhodou bezdrátových sítí je mobilita připojených zařízení neboť přenosovým médiem je u těchto sítí vzduch. Zařízení se tak mohou volně pohybovat a jsou omezena pouze dosahem signálu. Ten však závisí na mnoha faktorech jako je výkonnost vysílací antény nebo případné překážky mezi vysílačem a přijímačem.

Tím vzniká jeden z problémů bezdrátových sítí a tím je nespolehlivost přenosu informací. Neboť nelze zajistit přes jaké prostředí a na jakou dálku bude signál přenášen. Avšak se zlepšujícími se technologiemi se i tento problém stává méně znatelným.

Dalším problémem je komunikace všech zařízení pomocí jednoho přístupového bodu (access point, AP). Takže i když je dosaženo velké šířky pásma, připojená zařízení spolu soupeří o to, kdo zrovna bude komunikovat s přístupovým bodem. Tento problém se však snaží řešit nový standard WiFi 6 (IEEE 802.11ax).

Další problém vzniká interferencí signálů různých WiFi[1] sítí, případně i jiných rušivých elektromagnetických signálů. Moderní technologie se však i s tímto problémem poměrně dobře vypořádávají. Co však zůstává problémem je fakt, že signál je vyzařován do volného prostoru a může ho tedy odposlouchávat kdokoliv, kdo je v dosahu s patřičnou síťovou kartou. Z tohoto důvodu se komunikace mezi klientem a přístupovým bodem šifruje již na spojové vrstvě, aby případné odchycené pakety neměli žádnou vypovídající hodnotu.

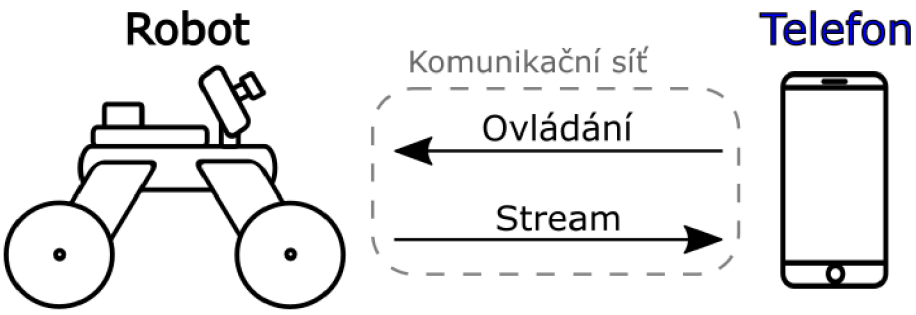
[1] WiFi – Wireless Fidelity, neboli WLAN – Wireless Local Area Network

6 Teoretický návrh

Tato kapitola se zabývá teoretickým návrhem aplikace pro pohybové ovládání robota a přenos obrazového signálu z webkamery robota. V aplikaci budou dva komunikační body. Jedním bude samotný robot a druhým bude aplikace na mobilním telefonu s operačním systémem Android. Aplikace na mobilním telefonu bude sloužit jako uživatelské rozhraní odkud bude uživatel moci interagovat s celou aplikací. Konkrétně pomocí této mobilní aplikace bude uživatel schopen ovládat robota a sledovat živý přenos videa. Robot pak bude zajišťovat funkcionalitu serveru a tedy bude zajišťovat 2 služby pro mobilní telefon a to pohybové ovládání a živý videopřenos. Každá služba bude mít přiřazený svůj vlastní port, díky kterému bude moci interagovat s mobilním telefonem.

Pro umožnění síťové komunikace mezi mobilním telefonem a robotem je potřeba, aby se oba komunikační body nacházeli ve stejné síti. To bude v naší aplikaci zajištěno 2 způsoby. Prvním je, že robot sám vytvoří bezdrátovou síť, ke které se mobilní telefon připojí. Druhým způsobem je připojení mobilního telefonu i robota do externí již existující bezdrátové sítě. V obou případech se robot s mobilním telefonem dostanou do stejné sítě a mohou spolu přes tuto síť komunikovat resp. mobilní telefon může přistupovat ke službám nabízených robotem.

Služba ovládání robota bude ve funkci klienta na straně robota. To znamená, že mobilní telefon bude zasílat ovládací příkazy popisující chtěný pohyb a robot bude pohyb realizuje. Naproti tomu při službě streamování bude robot poskytovat (odesílat) tok multimediálních dat mobilnímu telefonu a ten bude stream pouze přijímat a zobrazovat. Logické schéma návrhu aplikace je na obrázku 6.1.



Obr. 6.1: Návrh aplikace

7 Realizace aplikace

Tako část práce se bude zabývat samotnou realizací ovládání pohybu robota a přenosu živého videa z webkamery robota na obrazovku mobilního telefonu. Bude popsáno jak samotné technické řešení, tak i výhody a nevýhody volby tohoto řešení. Ukázán bude modul zabezpečení a jeho funkce v obou módech připojení robota k mobilnímu telefonu. Nakonec bude popsán i samotný výpočet úhlových rychlostí jednotlivých kol a jejich převod na řídící signál pro motor kontroléry.

7.1 Topologie sítě

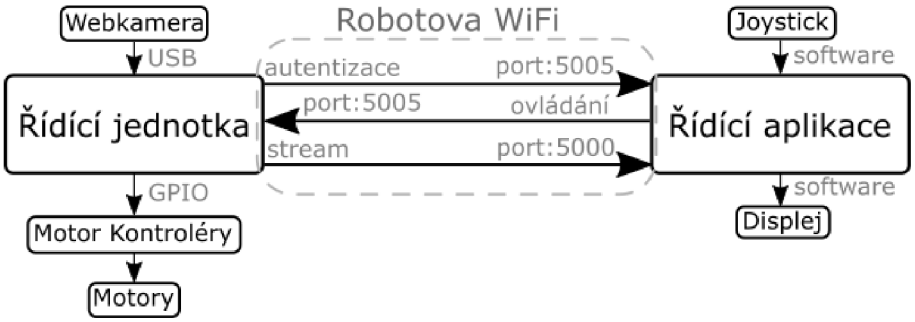
V této kapitole bude popsán způsob připojení robota k mobilnímu telefonu. Pro zajištění mobility robota je nutné použití bezdrátové komunikace. Konkrétně pak celá komunikace probíhá přes WiFi síť. Aplikace využívá 2 módy připojení. První mód, je mód vlastní WiFi sítě, ve kterém robot resp. řídící jednotka vytváří vlastní WiFi síť, na kterou se mobilní telefon připojí a následná komunikace probíhá na této vlastní síti. Druhý mód, je mód externí WiFi sítě, ve kterém se robot připojuje do externí WiFi sítě, kam se připojí i mobilní telefon a následná komunikace probíhá na této externí WiFi.

Více módový systém byl vytvořen z důvodu rozdílných charakteristik ovládání robota v obouch módech. Pro ovládání robota se tak vybere patřičný mód podle aktuální situace. Robot tak může využít výhod obou módů. Podrobnější popis jednotlivých módů, spolu s jejich specifikacemi, budou popsány v následujících podkapitolách.

7.1.1 Vlastní mód připojení

Mód vlastní WiFi sítě představuje situaci, kdy se Raspberry Pi stává přístupovým bodem k sebou vytvořené WiFi síti. Pro dosažení konektivity tak stačí, aby byl mobilní telefon připojen k této síti. Síť je přístupná pouze se znalostí přístupového hesla. Síť je lokální, bez přístupu k internetu. Slouží pouze pro připojení obou komunikačních bodů, robota a mobilního telefonu, do stejné sítě za účelem dosažení konektivity. Topologie tohoto módu je znázorněna na obrázku 7.1.

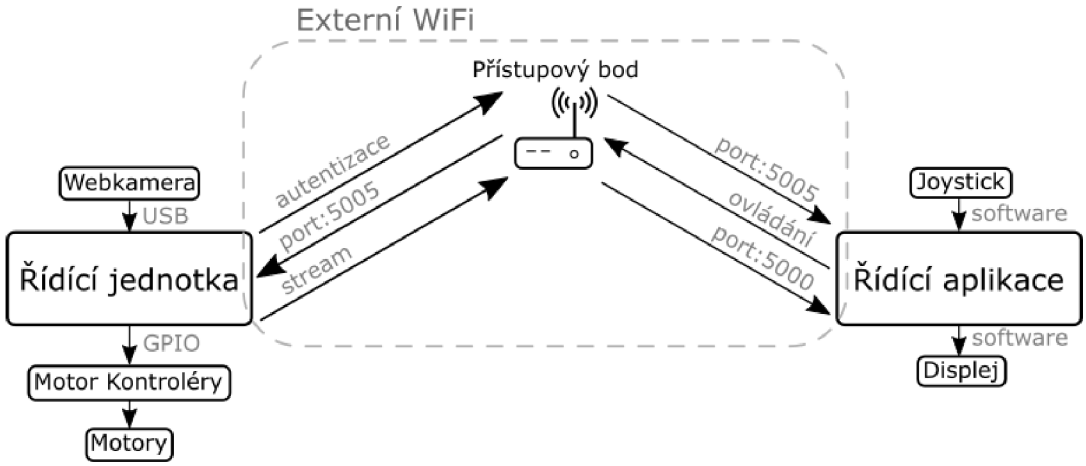
Síla WiFi signálu vysílaného Raspberry Pi není příliš vysoká a proto má robot velmi omezený dosah vzdálenosti, na kterou může být ovládán. Velkou výhodou tohoto módu však je nezávislost na externí infrastruktuře. Robot tak může potřebné prostředí pro jeho ovládání vytvořit kdekoliv, například i tam kde není dostupné připojení k internetu. Další výhodou je, že robot je správcem této sítě a může tak uzpůsobit síť podle potřeby.



Obr. 7.1: Topologie vlastního módu

7.1.2 Externí mód připojení

Mód externí WiFi sítě představuje situaci, kdy je robot připojen k externí WiFi síti, do které je připojen i mobilní telefon. Robot je na této síti pouze jedním z uživatelů dané sítě. To má za následek nemožnost zajištění nastavení sítě. Může se tak stát, že komunikace nebude šifrovaná nebo že na síti nebude místo, mobil nebo robot tak nemusí dostat přiřazenou IP[1] adresu a nebudou se tak moci připojit. Tato závislost na externí WiFi síti je největší nevýhodou tohoto módu. Většina externích přístupových bodů však mívá zpravidla silnější signál než Raspberry Pi. Se silnějším signálem se prodlužuje dosah ovládání robota. Větší dosah a jeho klavita je obvykle velkou výhodou tohoto módu. Topologie externího módu připojení je vidět na obrázku 7.2.



Obr. 7.2: Topologie externího módu

[1]Internet Protocol address – logická adresa komunikačního rozhraní počítače v dané síti – aplikace používá IPv4 adresy

7.1.3 Realizace módů připojení

Oba módy jsou realizovány za pomocí konfiguračních souborů na Raspberry Pi. Vlastní mód připojení je vytvořen za pomocí programu *hostapd* [17] a konfiguračního souboru *dhcpcd.conf*. Program *hostapd* umožní bezdrátovou síťovou kartu nastavit jako přístupový bod. Defaultní nastavení sítě je umístěno v souboru */etc/hostapd/hostapd.conf*. Nastavit lze například SSID[2], možnosti zabezpečení, heslo, kanál na kterém WiFi bude vysílat i typ modulace signálu tak jak jsou definovány ve standard IEEE 802.11. Dále je potřeba nastavení přidělování IP adres pomocí DHCP serveru. Jeho nastavení je v konfiguračním souboru */etc/dhcpcd.conf*. Po nastavení WiFi sítě je možné WiFi zapnou a vypnout pomocí služby *hostapd.service*. Pro přepnutí z externího módu do vlastního módu, slouží námi vytvořená python funkce *start_loc_wifi()*, která automaticky přepíše potřebné konfigurační soubory.

Externí mód připojení je zajištěn standardním konfiguračním souborem *wpa_supplicant.conf*, kde je možné nastavit SSID a heslo k WiFi, ke které se má Raspberry Pi připojit. Pro přechod z vlastního módu do externího tak stačí přepsat tento soubor a restartovat Raspberry Pi. Po restartu se Raspberry Pi samo připojí k nastavené WiFi. Automatizace tohoto procesu je zajištěna vytvořenou python funkcí *start_ext_wifi()*, která přijímá dva parametry SSID a heslo. Hodnoty těchto parametrů vpíše do konfiguračního souboru *wpa_supplicant.conf* a Raspberri Pi restartuje. Všechen zdrojový kód je dostupný v příloze.

7.2 Řídící aplikace

Řídící aplikace je označení pro Android aplikaci spuštěnou na mobilním telefonu, sloužící jako grafické uživatelské rozhraní (graphical user interface, GUI). Řídící aplikace umožňuje uživateli ovládání robota a sledování živého videopřenosu (streamu) z webkamery robota. Řídící aplikace je orientována naležato (landscape), převážně z důvodu co největší plochy pro zobrazení streamu a co nejpřirozenějšího rozložení komponent. Řídící aplikace je vytvořena v programovacím prostředí Android Studio a je z větší části napsána v programovacím jazyce Java a z menší části v programovacím jazyce C++. V jazyce Java je napsána hlavní funkcionalita řídící aplikace. Pomocí nástroje Android NDK je umožněna implementace části aplikace pomocí nativního kódu v jazyce C++ pro použití knihoven frameworku GStreamer [18] pro streamování videa. GStreamer, jeho funkce a použití v aplikaci budou popsány v oddělené kapitole 7.7 Streamování videa.

Pohybové ovládání robota je umožněno pomocí dvou virtuálních joysticků. Jeden v právem dolním rohu a druhý v levém dolním rohu. Joystick v pravém dolním rohu

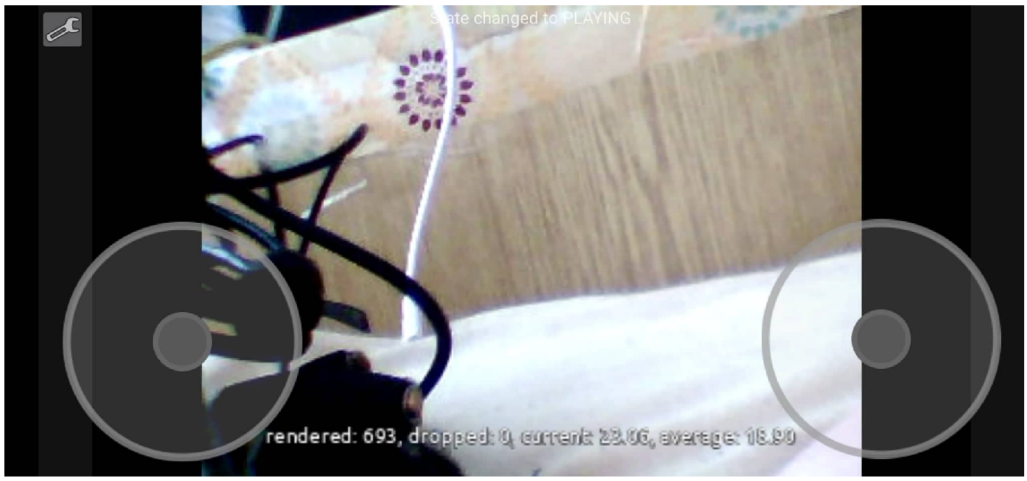
[2]Service Set IDentifier – pojmenování konkrátné WiFi sítě

ovládá všesměrový pohyb robota bez rotace a joystick v levém dolním rohu ovládá rotaci robota. Joystick v levém dolním rohu má i funkci Boost pro zrychlený přesun robota vpřed. Této funkcionalitě bude věnována speciální podkapitola 7.6.2 Boost.

Řídící aplikace má dvě aktivity – *Hlavní aktivita* a *Nastavení*. Hlavní aktivita slouží jako hlavní místo odkud může uživatel ovládat robota a Setting aktivita slouží pouze jako nastavení.

7.2.1 Hlavní aktivita

V této aktivitě může uživatel ovládat pohyby robota pomocí dvou virtuálních joystiků za současného sledování živého videopřenosu na pozadí. Tlačítkem vlevo nahoře má uživatel možnost přejít do nastavení. Vzhled a rozložení hlavní aktivity je vidět na obrázku 7.3.



Obr. 7.3: Hlavní aktivita

Hlavní aktivita obsahuje většinu funkcionality a je zde lokalizována většina kódu. V metodě *onCreate()*, která je volána při vzniku aktivity, jsou nejprve inicializovány všechny komponenty uživatelského rozhraní jako jsou tlačítka, joysticky, SurfaceView pro zobrazování video streamu a v neposlední řadě samotný GStreamer pro přijímání video streamu. Při jejich inicializaci se nastavují i event listenery, což jsou metody volané v případě, že proběhl event (událost) ke kterému jsou přiřazeny. Například pro setting tlačítko, tlačítko v levém horním rohu, sloužící pro přesměrování do Setting aktivity, je nastaven *OnClickListener()* u kterého je přepsána, pomocí anotace *@Override*, metoda *onClick()* ve které se nachází kód, jež bude vykonán při stisknutí daného tlačítka. Stejným způsobem je přepsanou metodou i výše popsaná metoda *onCreate()* volaná při zachycení eventu značícího prvotní vytváření aktivity. V těchto event listenerech jsou tedy naprogramovány funkcionality jednotlivých komponent uživatelského rozhraní. Funkce tlačítka a metody volané při jejim

stisknutí již byly vysvětleny a proto se nyní zaměříme na funkcionalitu virtuálních joystiků.

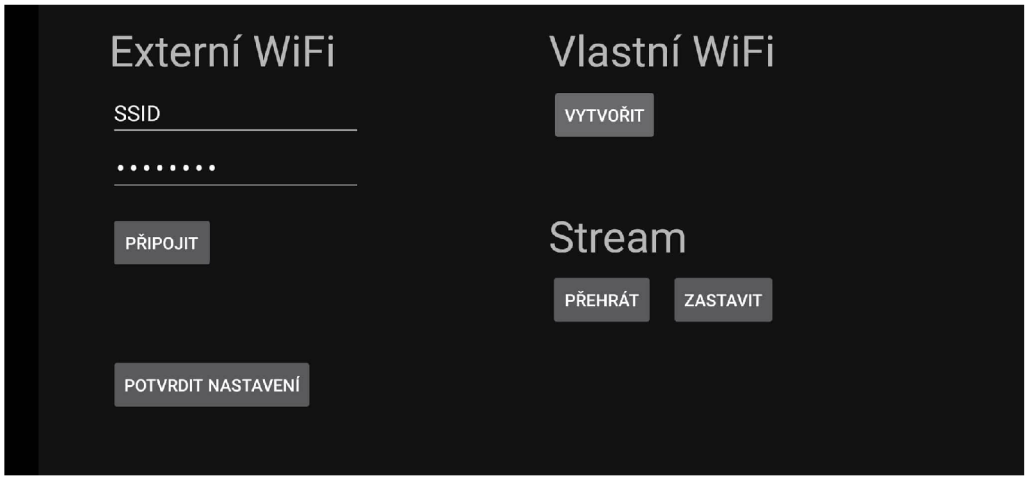
Virtuální joystick

Virtuální joystick není mezi standardními komponenty grafického rozhraní a proto je potřeba tuto specifickou komponentu nejprve vytvořit. Open source implementace této komponenty však již existuje a proto byla převzata z GitHubu [19]. Implementace je provedena v podobě Java třídy, která dědí z obecné třídy View a implementuje rozhraní Runnable pro práci ve vlastním vláknu. Operační systém Android je velmi citlivý na blokování hlavního vlákna aplikace. Po 5 sekundách blokace hlavního UI vlákna je aplikace operačním systémem vypnuta a všechny její prostředky jsou uvolněny. Proto se většina funkcionalit potenciálně nebezpečných z blokace hlavního vlákna implementují pro chod ve svém vlastním odděleném vlákně. Bude tomu tak například i u naší implementace UDP komunikace s robotem. Po importu JoystickView Java třídy do našeho projektu můžeme joystick používat. Byly přidány do hlavního layoutu což je prostor, kde se určuje rozložení komponent grafického uživatelského rozhraní. Po inicializaci v aktivitě v metodě *onCreate()*, již je joystick aktivní a po jeho dotyku je v pravidelných intervalech (defaultně 50ms) vyvoláván event *onMove()*. Ten můžeme, jak již bylo řečeno zachytit a přepsat metodu onMove() a určit tak reakci na aktuální pozici joystiku. K aktuální pozici je defaultně přístup pomocí úhlu ve stupních a procentuální vzdálenosti ovladače od středu. To pro naše použití není nejvhodnější a proto byl upraven zdrojový kód metody *onMove()* tak, aby jako parametr předávala X a Y pozici ovladače. Složením těchto souřadnic do vektoru pak vzniká směrový vektor $v = (x, y)$, který popisuje pohyb robota po ploše bez rotace. Vektor v spolu s rotací robota ω_z vytváří trojrozměrný vektor $m = (x, y, \omega_z)$, který plně popisuje pohyb robota. Tento vektor je zasílán přes síť pomocí protokolu UDP robotovi jako vstup pro realizaci chtěného ovládání pohybu. Tato funkcionalita bude podrobněji popsána v kapitole 7.6 Pohybové ovládání robota.

7.2.2 Nastavení

Nastavení celé aplikace je možné pomocí Setting aktivity. Jde, zde nastavit v jakém módu připojení chceme, aby robot operoval a spuštění nebo pozastavení služby streamování videa. Při volbě externího módu připojení je potřeba zadat SSID a heslo sítě, na které má robot operovat. Stiskem tlačítka „potvrdit nastavení“ je uživatel přesunut zpět na Hlavní aktivitu. Tato aktivita opět inicializuje všechny tlačítka a textová pole spolu s jejich funkcionalitou v event listenerech v metodě *onCreate()*,

volanou při vytvoření této aktivity. Vzhled a rozložení Setting aktivity je vidět na obrázku 7.4.



Obr. 7.4: Nastavení – Setting aktivita

7.2.3 Síťová komunikace

Komunikace mezi řídící aplikací a řídící jednotkou probíhá pomocí UDP datagramů. Operační systém Android nedovoluje použití hlavního UI vlákna pro práci se síťovou komunikací z důvodu možné blokace vlákna – čekání na příchozí datagramy. Proto byla vytvořena třída *UDP_Client*, která obsluhuje zasílání UDP datagramů a *UDPListenerService* služba pro přijímání datagramů v odděleném vlánkně.

Odesílání datagramů

UDP_Client třída má definovanou metodu *send()*, která odesílá zprávu, předanou v parametru, řídící aplikaci. Pro práci se sítí jsou použity interní Java třídy *DatagramSocket* a pak samotný *DatagramPacket*. Pro usnadnění práce metody ve vlastním vlákně je použitá třída *AsyncTask*, která nám pomáhá se správou prostředků pro vlákno a jeho bezkolizní běh. Metody využívající *AsyncTask* by měli ukončit svou práci do několika sekund a poté by jejích vlákno mělo být opět uvolněno. Hlavní aktivita tak může odesílat jakékoliv zprávy pomocí statické metody *send()*, není tedy ani potřeba vytvářet objekt *UDP_client třídy*. Obecně řídící aplikace zasílá 3 typy zpráv. První jsou HELLO zprávy při inicializaci IP adres. Druhý typ je digitální podpis při autentizaci. A třetím typem zpráv jsou zprávy pro samotné pohybové ovládání robota, obsahující vektor $m = (x, y, \omega_z)$ popisující chtěný pohyb robota.

Všechny tři typy zpráv a jejich bližší funkcionalita bude popsána v oddělených podkapitolách 7.4 Inicializace IP adres, 7.5.1 Autentizace řídící aplikace a 7.6 Pohybové ovládání robota.

Přijímání datagramů

Příjem datagramů znamená čekání dokud nějaký nepříjde. Po tuto dobu je vlákno blokováno. Metody pracující ve svém vláknu pomocí *AsyncTask* by měli mít dobu exekuce pár sekund. Proto nemůžeme použít *AsyncTask* pro přijímání datagramů a proto byla vytvořena speciální služba *UDPListenerService*, která běží také ve svém vlákně. Služby mohou běžet dokonce i když je hlavní UI vlákno na pozadí. Řídící aplikace tak může přijímat datagramy i pokud je aplikace minimalizovaná a uživatel se zrovna nachází v jiné aplikaci. Služba *UDPListenerService* naslouchá na portu 5005. Když na tento port přijde datagram, služba vyvolá námi definovanou událost (event) *MESSAGE_RECEIVED* a pomocí intentu předá obsah přijaté zprávy. Intent je objekt pomáhající pro přesun informací mezi jednotlivými komponenty jako jsou aktivity nebo služby. Na tento námi vyvolaný event čeká v hlavní aktivitě broadcast receiver, který zachytí danou událost spolu s jeho intentem. Tím se dostane Hlavní aktivita k obsahu každé přijaté zprávy a může na ni patřičně reagovat.

V našem případě jsou 3 možné typy přijatých zpráv, podobné dříve popsaným 3 odesílaným typům zpráv. Jsou jimi, inicializační HELLO zprávy, na které řídící aplikace odpovídá také zprávou HELLO. Autentizační zprávy obsahujicí *nonce* jako výzvu od robota pro ověření naši identity, na tyto zprávy řídící aplikace odpovídá digitálním podpisem daného nonce řetězce. A třetí typ zpráv jsou obyčejné informační zprávy zasílané robotem.

7.3 Řídící jednotka

Řídící jednotka je označení pro mikropočítač Raspberry Pi a na něm spuštěnou aplikaci serveru pro řízení robota a pro zprostředkování služby streamování videa. Je jedním z dvou komunikačních bodů celé aplikace, druhým je řídící aplikace. Hlavní funkcionalitou řídící jednotky je přijímání ovládacích zpráv od řídící aplikace a na základě těchto zpráv ovládat fyzický pohyb robota. Další funkcionalitou je odesílání živého videopřenosu z webkamery robota. V neposlední řadě řídící jednotka ověřuje identitu řídící aplikace, aby nedocházelo k neautorizovanému ovládání. Veškerou funkcionalitu zajišťuje python program jménem *UDP_server.py*. Program je spouštěn automaticky po zapnutí Raspberry Pi pomocí přidání *UDP_server.py* do konfiguračního souboru */etc/rc.local*. V obou módech má *UDP_server.py* stejný sled logických bloků. Prvně se musí inicializovat IP adresy na nové síti, pak autentizovat

řídící aplikaci na zjištěné adrese, následuje zahájení streamu pro řídící aplikaci na dané adrese a nakonec umožnění ovládání pohybu z dané řídící aplikace z konkrétní adresy. Pro samotné generování řídícího signálu, který je veden na motor kontroléry je použita knihovna gpiozero [20].

7.3.1 Síťová komunikace

Síťová komunikace ze strany řídící jednotky je realizována pomocí python modulu *Socket* umožňující práci se sockety. Pomocí tohoto modulu je vytvořen datagramový socket, kterému je následně přiřazen port 5005, na kterém řídící jednotka naslouchá. Po přijetí datagramu je obsah převeden, z jednotlivých bytů reprezentujících obsah zprávy, na řetězec znaků (string). Ten pak aplikace dále zpracovává.

7.4 Inicializace IP adres

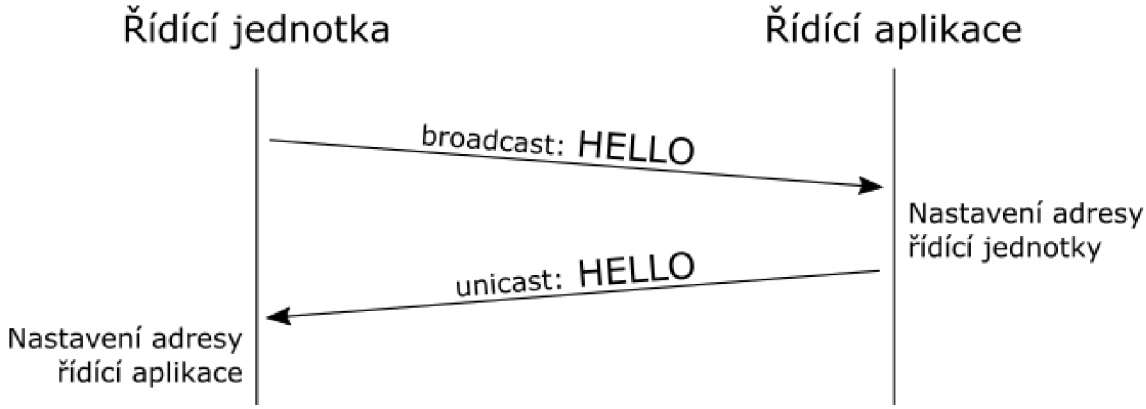
Pro unicastovou komunikaci mezi řídící aplikací a řídící jednotkou je potřebná znalost IP adres jednotlivých stran. Ty jsou však dynamicky přidělovány DHCP serverem. Proto je nejprve potřeba zjistit IP adresu protistrany. Tento problém řeší fáze inicializace IP adres na začátku každé komunikace.

Používají se takzvané HELLO datagramy, v této aplikaci sloužící pouze pro získání IP adres. Po tom, co se řídící jednotka přihlásí do externí sítě nebo vytvoří síť vlastní, začne vysílat periodicky každou sekundu všesměrový UDP datagram se všesměrovou cílovou adresou a řetězcem „HELLO“ jako obsah datagramu. Takto dává řídící jednotka všem v sítí vědět, že je přítomná a připravena ke komunikaci.

Ve chvíli, kdy takovýto HELLO datagram příjme řídící aplikace. Vyčte z hlavičky přijatého datagramu adresu řídící jednotky a uloží si ji. Zpět pak řídící aplikace odešle již unicastovou odpověď na již známou IP adresu řídící jednotky, také s obsaženým řetězcem „HELLO“. Řídící jednotka tento datagram příjme a také si z hlavičky vyčte a uloží IP adresu, tentokrát však adresu řídící aplikace. Tím jsou IP adresy inicializovány a může se pokračovat s dalším logickým blokem a tím je autentizace řídící aplikace. Průběh komunikace mezi řídící jednotkou a řídící aplikací při inicializaci IP adres je znázorněn na obrázku 7.5.

7.5 Modul zabezpečení

Hlavním úkolem bezpečnostního modulu je zajišťovat autentizaci řídící aplikace v obou módech připojení. V módu vlastní WiFi sítě, modul zabezpečení zajišťuje i šifrování komunikace. Obě funkcionality budou dále popsány.



Obr. 7.5: Inicializace IP adres

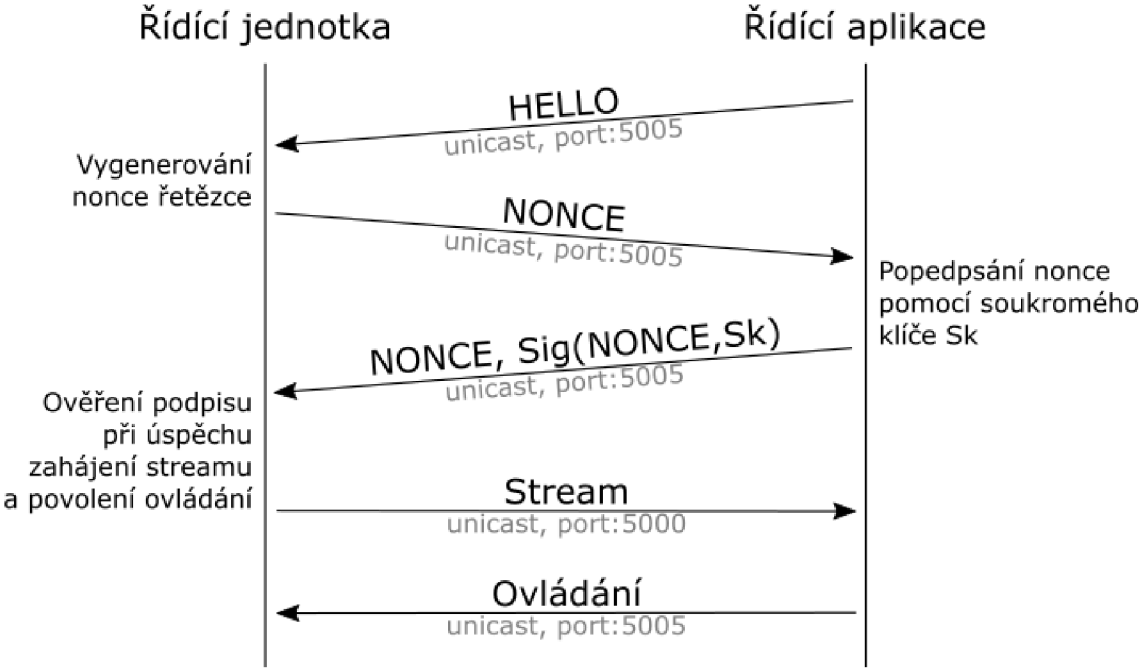
7.5.1 Autentizace řídící aplikace

Autentizace řídící aplikace je potřeba, neboť bez ní by řídící jednotka nevěděla s kým komunikuje a nedokázala by tak určit, kdo je oprávněný ovládat robota a kdo ne.

Řídící jednotka drží v paměti seznam aktuálně autentizovaných řídících aplikací na základě jejich IP adres. Ze začátku je tento seznam prázdný. Autentizace probíhá systémem výzva-odpověď s digitálním podpisem. Řídící jednotka tedy příjme žádost o autentizaci, v našem případě je to přijetí HELLO datagramu z inicializace IP adres, na kterou odpoví výzvou v podobě náhodně vygenerovaného řetězce nonce. V našem případě má nonce podobu náhodně vygenerovaného 27 místného čísla. Tento přijatý nonce řídící aplikace podepíše svým privátním klíčem a podpis odešle zpět řídící jednotce. Použitý algoritmus digitálního podpisu je RSA a použitá hash funkce je SHA-256. Řídící jednotka digitální podpis ověří odpovídajícím veřejným klíčem a v případě správného ověření podpisu, autentizace proběhla úspěšně a daná řídící aplikace je uložena do seznamu aktuálně autentizovaných řídících aplikací a program pokračuje do dalšího logického bloku (streamování a pohybové řízení). V případě špatného ověření digitálního podpisu je aplikaci přístup dále odepřen. Probíhající komunikace mezi řídící jednotkou a řídící aplikací v procesu autentizace je znázorněna na obrázku 7.6.

Všechny námi distribuované řídící aplikace, tak musí mít vygenerovaný asymetrický pár klíčů a mít uložen veřejný klíč z tohoto páru v paměti řídící jednotky. V aktuální verzi existuje pouze jedna řídící aplikace a je tedy uložen pouze její veřejný klíč.

Asymetrický pár klíčů je vygenerován v řídící aplikaci v Jave pomocí KeyPairGenerator třídy. Klíče jsou 2048 bitů dlouhé. Klíče jsou generovány v řídící aplikaci z důvod bezpečnosti, neboť se přenese pouze veřejný klíč, který není nijak utajovaný a privátní klíč není potřeba nikam přenášet a proto může být celou dobu v bezpečí uložen v řídící aplikaci, kde byl vytvořen. V řídící aplikaci jsou páry klíčů uloženy



Obr. 7.6: Autentizace

pomocí *Android KeyStore* systému [21], uložené pod aliasi a přístupné se znalostí hesla vytvořeného při jejich uložení. Veřejné klíče jsou v řídící jednotce uloženy ve formátu PEM[3].

7.5.2 Šifrování komunikace

Šifrování komunikace je zajištěno pouze ve vlastním módu připojení pouze za pomocí metody zabezpečení WPA2-PSK bezdrátové WiFi sítě. Šifrování je tak zajištěno na spojové vrstvě referenčního modelu ISO/OSI pro všechnu komunikaci odehrávající se na dané síti. Naopak jaké bude použito šifrování, případně jestli vůbec nějaké, v externím módu připojení nelze zajistit, už z povahy funkce robota jako pouhého uživatele externí WiFi sítě.

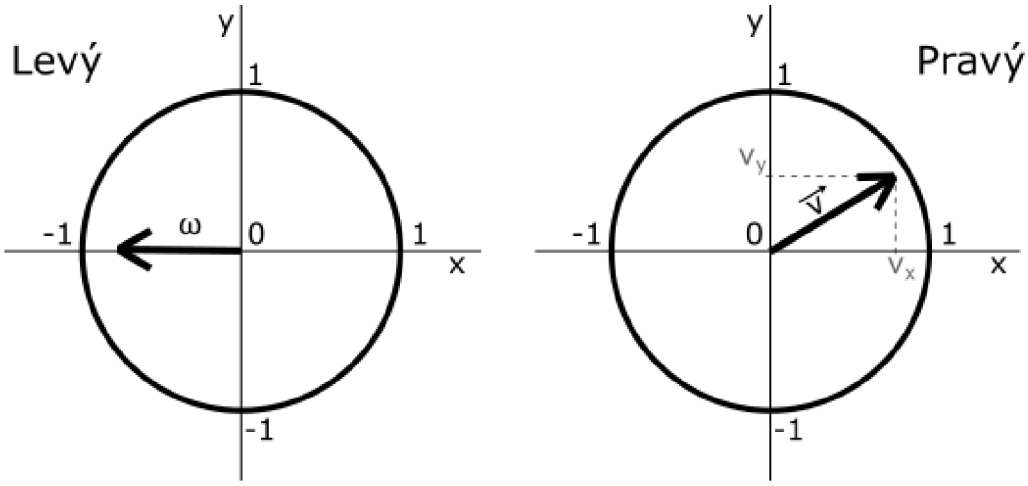
Jako budoucí vylepšení aplikace je možnost šifrování i na vyšších vrstvách. Tak by byla zajištěna důvěrnost komunikace i v externím módu a nehrozilo by prolomení šifrování zajištěného pomocí WPA2 [22]. Zasílané zprávy by mohli být šifrovány například pomocí AES algoritmu v bezpečném módu [23]. Tajný klíč by mohl být ustanoven například pomocí Diffie–Hellman protokolu.

[3]Privacy Enhanced Mail – speciální typ Base64 kódování, námi využit pro uložení veřejného klíče – obsahuje název použitého algoritmu, exponent a modulus

7.6 Pohybové ovládání robota

Funkce ovládání pohybu robota začíná v řídící aplikaci, kde skrze grafické uživatelské rozhraní, konkrétně pomocí dvou virtuálních joysticků může uživatel ovládat pohyby robota, tak jak bylo popsáno v druhém odstavci úvodu kapitoly *Řídící aplikace*.

Z virtuálního joysticku se každých 50 ms berou aktuální souřadnice joysticku. Virtuální joystick má 2 osy X a Y, je limitovaný jednotkovou kružnicí, počátky os jsou uprostřed této kružnice, kde se i joystik nachází v klidové poloze. Jeho dotykem ho lze vychýlit a po jeho puštění se opět vrací do počátku uprostřed kružnice se souřadnicemi $(0, 0)$. Rozsah jednotlivých souřadnic se pohybuje od -1 do 1. Obrázek 7.7 ukazuje levý a pravý joystick, jejich osy a proměnné které hodnoty reprezentují.



Obr. 7.7: Souřadnice levého a pravého joysticku

Pro pravý joystick, sloužící pro všesměrový pohyb bez rotace, se ze souřadnic vytvoří směrový vektor jízdy robota $v = (v_x, v_y) = (x, y)$. Vyjadřuje tak nejen směr, ale i velikost resp. rychlost chtěné jízdy robota. Hodnoty jsou vynásobeny hodnotou 0.2 pro lineární interpolaci[4] z rozsahu $[-1, 1]$ do rozsahu $[-0.2, 0.2]$ reprezentující rychlost v metrech za sekundu (m/s). U levého joystiku se využívána pouze osa X, neboť joystick je primárně – mimo funkce boost – využíván pro ovládání rotace robota a pro její určení stačí pouze jedna osa určující směr a velikost resp. rychlost otáčení. Rotace robota resp. úhlová rychlost ω_z robota kolem svislé osy Z, je vyjádřena hodnotou souřadnice osy X na levém joysticku. Pokud je hodnota záporná, robot rotuje vlevo a pokud je hodnota kladná robot rotuje vpravo, rychlostí danou nasledujícím převodem souřadnice na ose X. Souřadnice je násobena číslem 0.01 pro

[4]slouží pro převod (namapování) jednoho rozsahu na druhý, pomocí linearní rovnice, kde veličina z původního rozsahu představuje osu X a druhá veličina z druhého (namapovaného) rozsahu osu Y

lineární interpolaci z rozsahu $[-1, 1]$ do rozsahu $[-0.01, 0.01]$ reprezentující úhlovou rychlost robota kolem svislé osy Z v radiánech za sekundu (rad/s).

Ze složek směrového vektoru a úhlové rychlosti robota se vytvoří trojrozměrný vektor $m = (v_x, v_y, \omega_z)$, který plně popisuje pohyb robota. Řídící aplikace tento vektor odesílá řídící jednotce jako ovládací zprávu. Řídící jednotka tento vektor příjme a pomocí maticového výpočtu podle 3.1 vypočítá vektor obsahující úhlové rychlosti jednotlivých kol. Tyto úhlové rychlosti jsou převedeny do rozsahu 0 až 1, kdy 1 je maximální výkon daného kola. Takto převedené rozsahy jsou pak přivedeny na GPIO piny vyvedené na desce Raspberry Pi pomocí knihovny *gpiozero*. Signál z GPIO pinů je pak přiveden na motor kontroléry, které pomocí H-můstků regulují výkon na samotných motorech. Výpočet úhlové rychlosti a převod rozsahu bude blíže vysvětlen.

7.6.1 Výpočet úhlových rychlostí

Jako vstup pro výpočet úhlových rychlostí je trojrozměrný vektor $m = (v_x, v_y, \omega_z)$. Spolu se znalostí vzdálenosti středu kola od středu robota l_x, l_y a jeho poloměru r, můžeme pomocí vzorečku 3.1 vypočítat čtyřrozměrný vektor $(\omega_1, \omega_2, \omega_3, \omega_4)$, vektor s jednotlivými úhlovými rychlostmi pro jednotlivá kola. Pro generování řídícího signálu pro H-můstky je použit Python modul *gpiozero*, který má funkce *forward()* a *backward()*, které přijímají jako parametr číslo v rozsahu $[0, 1]$, kdy při 0 motor stojí a při 1 motor jede na plný výkon. Proto je potřeba jednotlivé úhlové rychlosti převést z rozsahu, který mají po výpočtu čtyřrozměrného vektoru, do rozsahu $[0, 1]$. Při řešení tohoto převodu, se objevují problémy – musíme počítat se specifickým všesměrovým charakterem pohybu robota. Problematiku si blíže popíšeme.

Problematika převodu

Všesměrový pohyb je zajištěn proměnlivou úhlovou rychlostí jednotlivých kol. Tím jde zajistit, že robot dokáže jet libovolným směrem po ploše a současně se otáčet. Změny v jednotlivých úhlových rychlostí a jejich dopad na pohyb robota jsou popsány v dřívější kapitole Kinematika Mecanum kol. Matematicky je tento vztah úhlových rychlostí kol a pohybu robota popsán maticí 3.1 a od ní odvozenými. Problém spočívá v tom, že matematická rovnice 3.1 jež používáme pro výpočet úhlových rychlostí kol, nebere v potaz, že naše motory mají limitovaný výkon. Je tedy naší povinností zajistit převod vypočtených matematických hodnot do hodnot limitovaných výkonem motoru. Pro tento převod použijeme lineární interpolaci. Musíme si však určit maximální úhlovou rychlost. Volba tohoto bodu změní charakter pohybu. Například, při zvolení nejjednoduššího způsobu a to dosazení za maximální úhlovou rychlost, rychlost robota při plném výkonu kol vpřed. Pak by nám po převodu již

nezbýval žádný dodatečný výkon potřebný při současném přidání například rotace. Rotace potřebuje, aby kola na jedné straně zpomalila a na druhé přidala. Avšak v našem případě, kdy robot jede rovně již na plný výkon motorů, tak nemáme výkon který by mohl být přidán na druhé straně kol. Může pouze ubrat rychlost kol na první straně, což rotaci zajistí avšak ne rychlostí kterou si uživatel zvolil. K dané situaci však uživatel může přidat ještě diagonální pohyb, který potřebuje přidat ještě další dodatečný výkon motorů a získáme tak pohyb robota který už výrazně neodpovídá očekávanému pohybu.

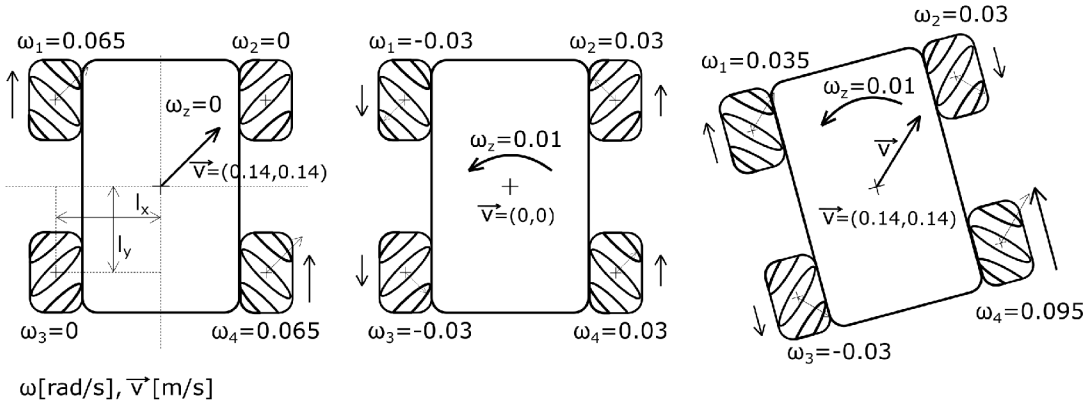
Řešení převodu

Pro dosažení nejpřirozenějšího chování pohybu robota musíme obětovat určitý výkon pro obyčejné pohyby jako je pohyb vpřed pro vytvoření dostatečného prostoru pro přidání dalších pohybů jako je pohyb diagonální a rotace. Jako maximální rychlost tak zvolíme maximální možnou úhlovou rychlost na jedno kolo. Tato rychlost je dosažena při diagonálním pohybu a současné rotaci. Samozřejmě při jejich maximálních hodnotách, které jsou sofwarově omezeny. Maximální velikost $|v_{max}|$ směrového vektoru $v = (v_x, v_y)$ a tedy odpovídající rychlost robota bez rotace je 0.2 m/s. Při této rychlosti v diagonálním směru a spolu s maximální rychlostí rotace robota 0.01 rad/s, je maximální úhlová rychlost na jedno kolo 0.095 rad/s. To je tedy maximální bod naší lineární interpolace.

Pro jednotlivé úhlové rychlosti kol tedy potřebujeme provést převod z rozsahu $[-0.095, 0.095]$ do rozsahu $[0, 1]$. To provedeme následovně. Záporné hodnoty budou volány s funkcí *backward()* a kladné hodnoty včetně 0 budou volány s funkcí *forward()*, dále tak můžeme brát pouze absolutní hodnoty a dostaneme se tím do rozsahu $[0, 0.095]$. Následně hodnoty vydělíme číslem 0.095 a tím je převod hotový a my získáváme rozsah $[0, 1]$ potřebný pro funkce *forward()* a *backward()* z modulu *gpiozero*.

Tímto tedy ztratíme výkon při nesložených pohybech. Například i rotace musela být omezena na řečených 0.01 rad/s, byť robot by zvládl takřka dvojnásobek. Ztracený výkon je však přeměněn do přirozenějších a plynulejších charakteristik ovládání robota. Pozitivem je také fakt, že i ovládání pomocí kamery je pohodlnější při nižších rychlostech robota. Nicméně abychom nepřišli zkrátka, tak byla vymyšlena a implementována funkcionalita Boost, která ztrátu výkonu, částečně kompenzuje.

Obrázek 7.8 popisuje zleva nejprve diagonální pohyb, pak pouze rotaci a pak kombinaci diagonálního pohybu s rotací, kde vzniká nejvyšší uhlová rychlost na jedno kolo. Pro všechny tyto situace jsou vypsány jejich směrové vektory v, rotace ω_z a jim odpovídající jednotlivé úhlové rychlosti pro jednotlivá kola.

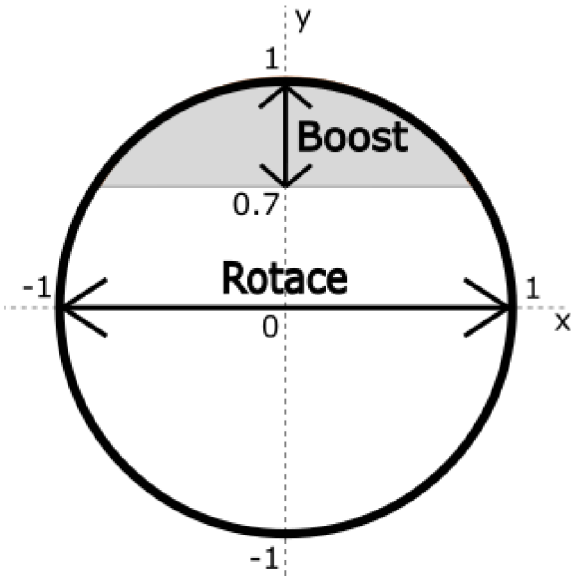


Obr. 7.8: Jednotlivé úhlové rychlosti kol

7.6.2 Boost

Funkcionalita Boost umožňuje využití plného výkonu motorů pro jízdu vpřed a minimalizovat tak dopad snížení výkonu při výpočtu a převodu úhlových rychlostí kol. Původně byla navržena funkcionalita Boost přístupná při stisknutí tlačítka. Tato varianta byla však nahrazena sofistikovanější variantou aktivace pomocí levého joysticku. Aktivace je tak pohodlnější a plynulejší. Pro aktivaci je použita doposud nevyužitá osa Y u levého joysticku. Osa X levého joysticku slouží pro ovládání rotace robota. Tomu tak i zůstane a bude to i nadále primární funkce levého joysticku. Pro využití funkce Boost bude využita pouze horní třetina joystiku resp. osy Y. Aktivace funkce Boost je možná posunutím levého joysticku do horní třetiny za současného plného výkonu vpřed na pravém joystiku. Tato podmínka byla přídána z důvodů plynulého navázání rychlostí. Oblast osy Y na levém joysticku, ve které je namapovaná funkcionalita Boost je vidět na obrázku 7.9.

Aktuální rychlost přímo vpřed bez funkce Boost je 0.2 m/s. Maximální možná rychlost jež je robot schopen s plným výkonem všech motorů je 0.25 m/s. Pravým joystickem jde tedy standardně ovládat pohyb robota od 0 až po 0.2 m/s. Funkcionalita boost přidává dodatečnou možnost řídit rychlost robota v rozsahu 0.2 až 0.25 m/s. Rozhas 0.7 až 1 osy Y je pomocí lineární interpolace namapován na hodnotu rychlosti robota vpřed, resp. je upravena velikost směrového vektoru v, na rozsah 0.2 až 0.25 m/s. Takto je zajištěno plynulé navýšení rychlosti až do maximální hodnoty.



Obr. 7.9: Namapování funkce boost

7.7 Streamování videa

Živý videopřenos z kamery robota je v naší aplikaci postaven na GStreameru. GStreamer je open source software [24] pod licencí LGPL[5]. GStreamer slouží pro sestavení cesty (roury) mezi zdrojem multimediálního obsahu, jako je kamera a zařízením, které tento obsah dokáže zobrazit, jako je například displej. GStreamer je tedy pipiline-based multimediální framework. Pipeline-based znamená, že fungování GStremeru je založeno na principu rour (pipelines). Z jednotlivých stavebních bloků (elementů) je vždy sestavena roura (potrubí), které spojí zdroj (source) multimediálního obsahu a zařízení pro zobrazení daného obsahu (sink). Zdrojem může být kamera, TV tuner nebo třeba ale i soubor či tok dat (stream) příchozí přes síť. Každý zdroj mívá své specifické formáty, které se nemusí shodovat s formáty podporovanými v zařízení pro zobrazení. Proto se pomocí GStreameru vytvoří roura s takovými elementy, které převedou daný tok dat do správného formátu, aby ho bylo možné zobrazit. Poskládáním jednotlivých elementů za sebe, se tedy vytvoří pipeline (roura) za účelem takového převodu. Roury jsou obsaženy v objektu jménem Bin. Jednotlivé elementy mají buď source (výstup), sink (vstup) nebo oboje, podle toho jakou funkci plní.

[5]GNU Lesser General Public License – je free software licence primárně používaná pro vytváření knihoven

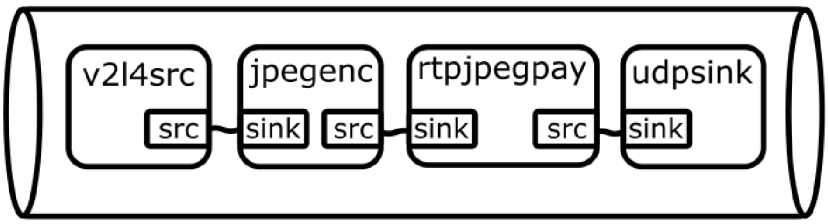
Pro přenos živého videa z webkamery robota na displej mobilního telefonu byly sestaveny roury dvě. Jedna na řídící jednotce, která má jako zdroj webkameru a končí elementem který vytváří RTP stream (Real-time Transport Protocol). Druhá roura je pak vytvořena v řídící aplikaci, která má příchozí RTP stream jako zdroj a končí předáním streamu Android komponentě, která ho zobrazí uživateli. Sestavení jednotlivých rour je dále blíže popsáno.

7.7.1 Roura na řídící jednotce

Pro sestavení roury na řídící jednotce byl použit nástroj gst-launch-1.0. Přikaz pro sestavení této roury vypadá následovně:

```
v4l2src ! video/x-raw,format=YUY2,width=320,height=240 ! jpegenc !
rtpjpegpay ! udpsink host=<dest_IP_addr> port=<dest_port>
```

Jako první je element *v4l2src*, který je využit jako source (zdroj) pro získání videa z v4l2 zařízení na Rasppbery Pi. V4l2 je kolekce ovladačů a vytvořeného rozhraní pro práci s realtime videem v operačních systémech linux. Jako rozhraní je vytvořeno virtuální v4l2 zařízení, kterému je pak přiřazeno reálně připojené zařízení. V našem případě tedy v4l2 odkazuje na webkameru připojenou pomocí USB. Dále je specifikovaný formát výstupu *v4l2src* elementu a to na kódování YUY2 s rozlišením 320x240 pixelů. Následně *jpegenc* element převede formát na JPEG a posílá ho elementu *rtpjpegpay*, který vytvoří jednotlivé RTP pakety. Z těchto paketů nakonec *udpsink* vytváří UDP datagramy, které posílá po síti na specifikovanou adresu a port. Tato roura je graficky vyobrazena na obrázku 7.10.



Obr. 7.10: Roura pro odesílání streamu

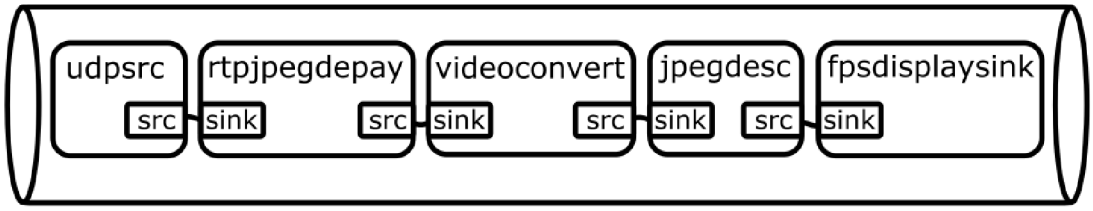
7.7.2 Roura na řídící aplikaci

V řídící aplikaci je pak potřeba sestavit rouru, která bude přijímat tento RTP stream. Pro sestavení roury již nelze použít nástroj gst-launch-1.0 a proto musíme rouru sestrojit sami pomocí jazyka C++. Abychom mohli použít tento jazyk, při vytváření Android aplikací, použijeme soubor nástrojů Android NDK. Po jeho přiřazení do

projektu a vytvoření *Android.mk* spolu s *Application.mk* pro zajištění správné kompilace kódu napsaného v souboru *gst_lib.cpp*, Android Studio samo vytvoří APK soubor s naší aplikací obsahující zkompilovanou knihovnu *gst_lib.so*, ze které může Java volat napsané funkce. *Android.mk*, *Application.mk* a *gst_lib.cpp* jsou uloženy v souboru */jni*. Na jejich obsah je možné se podívat v příloze. Roura je sestavena při vykonávání hlavní funkce *app_function()*. Sestavená roura pro přijímání streamu vypadá následovně:

```
udpsrc port=5000 ! application/x-rtp,media=video,payload=26,
clock-rate=90000,encoding-name=JPEG,framerate=30/1 ! rtpjpegdepay !
jpegdec ! videoconvert ! fpsdisplaysink video-sink=autovideosink
```

Jako zdroj je zde element *udpsrc*, který přijímá přicházející UDP datagramy na portu 5000. Tyto RTP datagramy jsou předány elementu *rtpjpegdepay* a ten je převede na JPEG video. To je dekódováno pomocí elementu *jpegdec* a videoconvertu. Nakonec je pomocí *fpsdiplaysink* video předáno samotné řídící aplikaci. Dále je už pouze přijatý stream vykreslen na displej telefonu. Implementace Gstremeru do řídící apliakce byla vytvořena s ohledem na oficiální ukázkové příklady použití GStremeru v Android aplikacích [25]. Na obrázku 7.11 je graficky vyobrazena roura pro příjem RTP streamu v řídící aplikaci.



Obr. 7.11: Roura pro příjem streamu

Pro zobrazení streamu byla použita Android komponenta grafického uživatelského rozhraní *SurfaceView*. Avšak pro definování lépe přizpůsobivého chování, jako například změna velikosti podle šířky a výšky aktuálně používaného telefonu, byla vytvořena vlastní komponenta dědící právě ze *SurfaceView*. Námi definovaná komponenta byla pojmenována *GstreamSurfaceView*, dědí ze *SurfaceView* a přepisuje její původní metodu *onMeasure()*, právě pro definování responzivního chování velikosti zobrazovací plochy. Tato komponenta je přidána na pozadí hlavní aktivity, kde se nachází i ovládací prvky. V již dříve přepsané metodě *onCreate()* v aktivitě je nyní volaná inicializace GStreamu, poté je volána samotná funkce pro vytvoření roury, přijímání streamu a následně je přijímaný obsah zobrazován na zmíněném *GstreamerSurfaceView* umístěn na pozadí hlavní aktivity. Dále jsou v C++ kódu

definovány funkce pro pozastavení s opětovné spuštění streamu. Tyto funkce jsou volány při stisknutí odpovídajících tláčítek v Setting aktivitě.

8 Testování aplikace

Robota lze poměrně pohodlně ovládat pomocí virtuálních joystiků jak s přímým pohledem na robota, tak bez přímého pohledu pouze pomocí jeho webkamery. Robot překonává i poměrně nerovnoměrný povrch, avšak všesměrová charakteristika pohybu Mecanum kol vynikne nejlépe na hladkém a přilnavém povrchu. Videopřenos byl po většinu provozu plynulý a takřka bez zpoždění. Tyto popsané charakteristiky však platily pouze při silném či lehce slabším signálu WiFi sítě. Při přesunu robota do oblasti s horším signálem nastává bod, kdy se skokově zhoršuje kvalita, jak streamu tak ovládání. Zhoršení se projevuje sekaným pohybem přijímaného streamu a velkého zpoždění (1-2s) streamu i ovládání. V takové situaci je nejlepší robota vzít a ručně přenést zpět do oblasti s lepším signálem. Dá se však i bez externí pomoci s robotem dojet zpět do takové oblasti, pouze se robot musí ovládat krátkými impulzy a čekat na odezvu robota, ať už při přímém dohledu nebo pomocí kamery. Pomocí kamery však proces trvá značně delší dobu.

8.1 Vlastní mód připojení

Při ovládání robota ve vlastním módu připojení se robot dostává do oblasti špatného signálu již při poměrně malých překážkách mezi robotem a řídící aplikací, stačí například i kuchyňská linka. Robot je tak poměrně neovladatelný pouze pomocí jeho webkamery. Respektive ovládání pomocí kamery je bezproblémové při přímém dohledu na robota. V takovém případě však možnost řízení pomocí kamery není tak užitečná.

8.2 Externí mód připojení

V externím módu připojení se občas objevuje lehce vyšší zpoždění, ale i tak je zpoždění takřka neznatelné při silném signálu. V tomto módu je dominantou řízení robota pomocí kamery. Dosah sítě při testování dosahoval takřka po celém domě. Robota tak šlo ovládat bez problému pouze za pomocí kamery i když byl robot v jiné místnosti. Opět ale v místnostech se slabším signálem nastávali problémy s vyšší odezvou.

8.3 Přepínání mezi módy

Robot se při zapnutí nacházel ve vlastním módu. Po připojení telefonu k vytvořené WiFi se znalostí hesla, bylo možné ovládat robota v podstatě okamžitě. Proces ini-

cializace a autentizace proběhl do zlomku sekundy. Pro změnu módu byly zadány přihlašovací údaje externí WiFi sítě. Robotem vytvářená síť okamžitě zmizela a telefon se automaticky připojil k zadané externí síti. Bylo tomu tak, neboť externí síť byla domácí sít, na které telefon standardně operuje, a proto se opět telefon automaticky připojil, po ztrátě WiFi od robota. Doba potřebná, než se robot také připojil a bylo možné robota opět ovládat, byla kolem 1 minuty. Poté šlo robota opět ovládat. V testovacím prostředí byl dobrý signál v obou módech a tak nebyl citelný rozdíl mezi ovládáním v jednom nebo druhém módu. Pro přepojení zpět do vlastního módu stačilo stisknout tlačko pro vlastní mód a opět zhruba po 1 minutě se objevila robotem vytvořená WiFi síť, k jejímu připojení již stačilo kliknout na dané SSID bez nutnosti znovu zadávat heslo (bylo telefonem zapamatováno) a ovládání bylo opět v provozu.

Závěr

Zadání bakalářské práce, návrh a realizace aplikace pro směrové ovládání pohybu robota a zobrazení obrazového signálu z kamery, bylo splněno. V práci byl nejprve popsán robot a změny na něm provedené. Poté byly popsány různé způsoby pohybového ovládání robota pomocí dotykové obrazovky, jejich výhody a nevýhody a byl vybrán způsob pomocí virtuálních joystiků. Byla vytvořena řídící Android aplikace jako uživatelské rozhraní pro ovládání robota. Napsána byla i Python aplikace pro samotné ovládání robota a zprostředkování živého videopřenosu z webkamery robota. Pro připojení robota s telefonem do stejné sítě byly použity dva módy. Vlastní mód s vytvořením vlastní WiFi sítě a externí mód pro připojení do již vytvořené WiFi sítě. V uživatelské Android aplikaci bylo vytvořeno nastavení, ve kterém si uživatel může vybrat mód připojení a spuštění či pozastavení služby zobrazování živého videopřenosu. Nakonec byla celá aplikace otestována.

Jako budoucí rozšíření aplikace by bylo vhodné přidání možnost nastavení rozlišení přenášeného streamu, přidání dodatečného šifrování komunikace na aplikační vrstvě, autentizace i ze strany robota, výměna webkamery za kvalitnější, přidání mikrofonu pro přenos živého audia, případně zlepšení fyzické konstrukce robota pro menší otřesy či lepší uspořádání jednotlivých hardwarových komponent.

Literatura

[1] DVOŘÁČEK, Štěpán. *Robot s autonomním audio-vizuálním řízením.* Brno, 2019. Dostupné také z: https://www.vutbr.cz/studenti/zav-prace/detail/118147. Diplomová práce. Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav telekomunikací. Vedoucí práce Jiří Přinosil.

[2] Pololu Robotics and Electronics: 120:1 Mini Plastic Gearmotor, 90° 3mm D-Shaft Output [online]. Las Vegas, Nevada: Pololu Corporation, 2001 [cit. 2020-11-29]. Dostupné z: https://www.pololu.com/product/1124

[3] DRV8835 Dual Motor Driver Carrier [online]. 920 Pilot Rd. Las Vegas, NV 89119 USA: Pololu Corporation [cit. 2021-04-15]. Dostupné z: https://www.pololu.com/product/2135

[4] Raspberry Pi 4 Model B [online]. 37 Hills Rd, Cambridge CB2 1NF, United Kingdom: Raspberry Pi Foundation [cit. 2021-04-15]. Dostupné z: https://www.raspberrypi.org/products/raspberry-pi-4-model-b/

[5] GEETHA, Ganesan a Rajeev SOBTI. *Cryptographic Hash Functions: A Review* [online]. School of Computer Science, Lovely Professional University Phagwara, Punjab 144806, India: International Journal of Computer Science Issues Vol 9, 2012. [cit. 2021-04-15]. Dostupné také z: https://www.researchgate.net/publication/267422045_Cryptographic_Hash_Functions_A_Review

[6] BURDA, Karel. *Aplikovaná kryptografie.* Brno: VUTIUM, 2013. ISBN 978-802-1446-120.

[7] ILON, Bengt. *heels for a course stable selfpropelling vehicle movable in any desired direction on the ground or some other base.* US3876255A. Uděleno 04/08/1975. Zapsáno 11/13/1972.

[8] BLACKWELL, Mike. *The URANUS Mobile Robot* [online]. Pittsburgh: Camegie Mellon University, 1990 [cit. 2020-11-29]. Dostupné také z: https://www.ri.cmu.edu/pub_files/pub3/blackwell_mike_1990_1/blackwell_mike_1990_1.pdf

[9] DOROFTEI, Ioan, Victor GROSU a Veaceslav SPINU. *Omnidirectional Mobile Robot - Design and Implementation* [online]. Technical University of Iasi. Romania, 2007 [cit. 2020-11-29]. Dostupné z: https://www.intechopen.com/books/bioinspiration_and_robotics_walking_and_climbing_robots/omnidirectional_mobile_robot_-__design_and_implementation

[10] TAHER, Hamid, Bing QIAO a Nurallah GHAEMINEZHAD. *Kinematic Model of a Four Mecanum Wheeled Mobile*. 2015, March 2015(3). Dostupné také z: https://research.ijcaonline.org/volume113/number3/pxc3901586.pdf

[11] Electric wheel chair [online]. EVER Monaco 2006 fair for mobility with renewable energy: Planetary Engineering Group Earth, 2006 [cit. 2020-11-29]. Dostupné z: https://car.pege.org/2006-ever-monaco/wheel-chair.htm

[12] Android documentation. Adnroid Developers [online]. Google [cit. 2021-04-15]. Dostupné z: https://developer.android.com/docs

[13] ROGERS, Rick, et al. *Android Application Development*. O'Reilly Media, 2009. ISBN 9780596521479.

[14] Android software development, 2001-. In: Wikipedia: the free encyclopedia [online]. San Francisco (CA): Wikimedia Foundation [cit. 2020-11-28]. Dostupné z: https://en.wikipedia.org/wiki/Android_software_development

[15] Android NDK. Adnroid Developers [online]. Google [cit. 2021-04-15]. Dostupné z: https://developer.android.com/ndk/guides

[16] KLEMENT, Milan. Technologie bezdrátových sítí základní principy a standardy. Univerzita Palackého v Olomouci, 2017. ISBN 9788024451565.

[17] JOUNI, Malinen. Hostapd documentation [online]. 2002 [cit. 2021-04-15]. Dostupné z: https://w1.fi/hostapd/

[18] GStreamer [online]. GStreamer Team [cit. 2021-04-15]. Dostupné z: https://gstreamer.freedesktop.org/documentation/index.html?gi-language=c

[19] BRUN, Damien. *Virtual joystick android* [online]. GitHub [cit. 2021-04-15]. Dostupné z: https://github.com/controlwear/virtual-joystick-android

[20] Gpiozero [online]. Ben Nuttall, 2015 [cit. 2020-11-29]. Dostupné z: https://gpiozero.readthedocs.io/en/stable/api_output.html#motor

[21] Android KeyStore system. Adnroid Developers [online]. [cit. 2021-04-20]. Dostupné z: https://developer.android.com/training/articles/keystore

[22] SINGH, Rajiv Ranjan, José MOREIRA, Tom CHOTHIA a Mark D. RYAN. *Modelling of 802.11 4-Way Handshake Attacks and Analysis of Security Properties* [online]. Birmingham, UK: School of Computer ScienceUniversity of Birmingham, 2020 [cit. 2021-04-15]. Dostupné z: https://link.springer.com/chapter/10.1007%2F978-3-030-59817-4_1#CR20

[23] KLÍMA, Vlastimil. Základy moderní kryptologie - Symetrická kryptografie III: operační mody blokových šifer a hašovací funkce. 2005. Dostupné také z: http://crypto-world.info/klima/mffuk/Symetricka_kryptografie_III_2005.pdf

[24] GStreamer [online]. GStreamer team, GitLab [cit. 2021-04-15]. Dostupné z: https://gitlab.freedesktop.org/gstreamer/gstreamer

[25] GStreamer tutotials [online]. GStreamer team [cit. 2021-04-15]. Dostupné z: https://gstreamer.freedesktop.org/documentation/tutorials/android/index.html?gi-language=c

Seznam symbolů a zkratek

RC	Radio Controlled – dálkově ovládaný
DC	Direct current – stejnosměrný proud
PWM	Pulse-width modulation – pulzně šířková modulace
GPIO	General-purpose input/out-put
USB	Universal Serial Bus
MitM	Man in the Middle – muž uprostřed
OS	Operating system – operační systém
UI	User Interface – uživatelské rozhraní
NDK	Nativ Development Kit – balíček pro práci s nativním kódem
JNI	Java Native Iterface – rozhraní mezi Java a nativním kódem
AP	Access point – přístupový bod
GUI	Graphical User Iterface – grafické uživatelské rozhraní
LGPL	GNU Lasser General Public License – free software licence
RTP	Real-time Pransport Protocol – protokol pro přenost multimédií

A Obsah elektronické přílohy

Příloha obashuje dva adresáře – *android* a *python*. V adresáři *android* je možné nalést kód týkající se řídící aplikace. Naproti tomu v adresáři *python* lze nalést kód řídící jednotky. Adresář *android* obsahuje i C++ kód přidaný pomocí Android NDK i obrázky (sprites) použité pro vizualizaci joysticku.

```
/ ..... kořenový adresář přiloženého archivu
├── android ..... adresář s kódem řídící aplikace
│   ├── jni ..... adresář obsahující C++ kód a makefiles
│   │   ├── Android.mk
│   │   ├── Application.mk
│   │   └── gst_lib.cpp
│   └── main ..... adresář obsahující JAVA kód samotné Android aplikace
│       ├── Assets
│       ├── java
│       │   ├── cz.vutbr.controlapp ..... package s kódem
│       │   │   ├── CryptoManager.java
│       │   │   ├── GStreamerSurfaceView.java
│       │   │   ├── JoystickView.java
│       │   │   ├── MainActivity.java
│       │   │   ├── SettingActivity.java
│       │   │   ├── SignatureManager.java
│       │   │   ├── UDP_Client.java
│       │   │   └── UDPListenerService.java
│       │   └── org.freedesktop.gstreamer ..... knihovny GStreamer
│       ├── res
│       │   ├── drawable ..... adresář se spritem pro joystick
│       │   │   ├── joystick_background.png
│       │   │   └── joystick_button.png
│       │   └── layout ..... adresář s rozložením aktivit
│       │       ├── main.xml
│       │       └── setting.xml
│       └── AndroidManifest.xml
└── python ..... adresář s kódem řídící jednotky
    ├── UDP_server.py ..... python skript běžící na řídící jednotce
    ├── functions.py ..... pomocné funkce
    └── send.sh ..... bash skript pro zahájení streamu
```