ČESKÉ VYSOKÉ
UČENÍ TECHNICKÉ
V PRAZE

FAKULTA
STROJNÍ

ZÁVĚREČNÁ
PRÁCE
2019

ELISEYKIN
MAXIM

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Eliseykin** Jméno: **Maxim** Osobní číslo: **439314**

Fakulta/ústav: **Fakulta strojní**

Zadávající katedra/ústav: **Ústav přístrojové a řídící techniky**

Studijní program: **Strojírenství**

Studijní obor: **Informační a automatizační technika**

II. ÚDAJE K BAKALÁŘSKÉ PRÁCI

Název bakalářské práce:

Mobilní aplikace pro bezdrátové řízení robota

Název bakalářské práce anglicky:

Mobile application for wireless robot control

Pokyny pro vypracování:

1) Zpracujte rešerši na téma: 'Mobilní aplikace v průmyslu a jejich použití, aktuální dostupnost pro hobby sektor'.
2) Vytvořte vlastní mobilní aplikaci bezdrátové řízení laboratorního robota. Zvažte, zda je možné aplikaci navrhnout tak, aby byla modulární a bylo možné ji jednoduše rozšiřovat o další akční prvky, např. pokud robot získá navíc další servo.
3) Aplikaci otestujte a navrhněte další možná rozšíření a použití.

Seznam doporučené literatury:

[1] WOLBER, David. App Inventor: create your own Android apps. Sebastopol, Calif.: O'Reilly, 2011. ISBN 14-493-9748-4.
[2] DUFFY, Thomas J. Programming with mobile applications: Android, iOS, and Windows Phone 7. Boston, MA: Course Technology/Cengage Learning, 2013. ISBN 11-336-2813-3
[3] Horton, John. Android programming for beginners, Packt Publishing , 2015, ISBN-13: 978-1785883262

Jméno a pracoviště vedoucí(ho) bakalářské práce:

Ing. Zdeněk Novák, U12110.1

Jméno a pracoviště druhé(ho) vedoucí(ho) nebo konzultanta(ky) bakalářské práce:

Datum zadání bakalářské práce: **26.04.2019** Termín odevzdání bakalářské práce: **12.06.2019**

Platnost zadání bakalářské práce: _____

Ing. Zdeněk Novák podpis vedoucí(ho) práce	podpis vedoucí(ho) ústavu/katedry	prof. Ing. Michael Valášek, DrSc. podpis děkana(ky)

III. PŘEVZETÍ ZADÁNÍ

Student bere na vědomí, že je povinen vypracovat bakalářskou práci samostatně, bez cizí pomoci, s výjimkou poskytnutých konzultací. Seznam použité literatury, jiných pramenů a jmen konzultantů je třeba uvést v bakalářské práci.

26.4.2019
Datum převzetí zadání

Podpis studenta

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně s tím, že její výsledky mohou být dále použity podle uvážení vedoucího diplomové práce jako jejího spoluautora. Souhlasím také s případnou publikací výsledků diplomové práce nebo její podstatné části, pokud budu uveden jako její spoluautor.

Dne ... Podpis ...

Poděkování

Chtěl bych poděkovat ústavu přístrojové a řídicí techniky za vytvoření kvalitních podmínek pro vypracování bakalářské práce. Můj velký dík patří Ing. Zdeňku Novákovi za předání zkušeností, cenné rady, a za odborné vedení při vypracování bakalářské práce. Stejně tak děkuji i všem pedagogům FS ČVUT v Praze.

Abstract

Cílem této bakalářské práce je navrhnout a otestovat mobilní aplikaci pro bezdrátové řízení laboratorního robota.

První bod tvoří úvod do automatizovaných systémů řízení technologických procesů a rešerši na téma mobilní aplikace v průmyslu a hobby sektoru. Druhý bod pojednává o výběru vhodné platformy a programovacím jazyku, návrhu struktury programu a kódování. Ve třetím bodě je cílem otestovat a doladit aplikaci a navrhnout další rozšíření a možné použití. Byl úspěšně zprovozněn podvozek robota, na kterém byla aplikace otestována. Při testování vznikly pouze zanedbatelné chyby, hardware správně interpretuje přijímaná data a převádí je do signálů pro řízení robota. Navrhnutou aplikaci lze modifikovat a rozšířit pro použití v dalších úlohách.

Klíčová slova:

Průmyslové řídící systémy, mobilní aplikace, Android, bezdrátové řízení, Java

The aim of this bachelor thesis is to design and test mobile applications for wireless laboratory robot control.

The first point is an introduction to automated process control systems and a search for mobile application in the industry and hobby sector. In the second point, select the appropriate platform and programming language, design the program structure and encode. At the third point, test and fine-tune the application and suggest further extensions and used. The robot chassis was put into operation and the application tested on the chassis was not too critical of errors, the hardware correctly interprets the received data and converts it into robot control signals. The designed application can be modified and extended to be used in other tasks.

Keywords:

Automated Control Systems, Mobile Application, Android, Wireless Control, Java

Obsah

Úvod

V současné době se mobilní technologie staly nedílnou součásti našeho života, a to nejen v domácnosti ale i v průmyslu. Velké rozšíření našly také mobilní aplikace, díky kterým jsme schopni mnohem rychleji a pohodlněji ovládat a spravovat obrovské množství různých procesů včetně těch výrobních. Avšak tvorba těchto aplikací není tak jednoduchá a je potřeba získat znalosti nejen z oboru pro které je aplikace navrhována, ale i z programovaní, designu a řízení.

Bezdrátové technologie také nezůstaly pozadu a představují jednu z nejrozvinutějších a nejslibnějších technologií. V dnešní době existuje několik klasifikací a standardů bezdrátových komunikací, které se rozlišují na různé rychlosti a možné dosahy. S jejich pomocí je možné se zbavit fyzických vodičů, které usnadní strukturování a revitalizaci továren, budov apod. Často také přinesou nezanedbatelné ekonomické úspory.

V poslední době je ze strany průmyslu velký zájem o téma mobilních řešení, který se navzdory svému konzervatismu snaží uplatňovat nové technologické nástroje ke zlepšení procesů. Též bankovní sektor a telekomunikace nachází stále větší uplatnění pro zmíněné technologie. Mnoho technologií, které byly dříve považovány za experimentální, jsou dnes už připraveny k implementaci.

Stroje s vestavěnými analytickými senzory, které jsou schopny komunikovat s externími zařízeními pomocí protokolů Bluetooth, LoRaWAN, NFC, se stávají de facto standardem ve výrobních zařízeních. Pokud neexistují, pak se v současné době rozvíjí obrovský trh senzorů, se kterými lze přizpůsobit stávající zařízení. Je možné přesně určit pravděpodobnost selhání strojů, optimalizovat jejich zátěž a nejzajímavější je vytvořit samostatné výrobní řetězce několika strojů, které si navzájem vyměňují informace, pracující bez lidského zásahu.

1 Mobilní aplikace v průmyslu a hobby sektoru

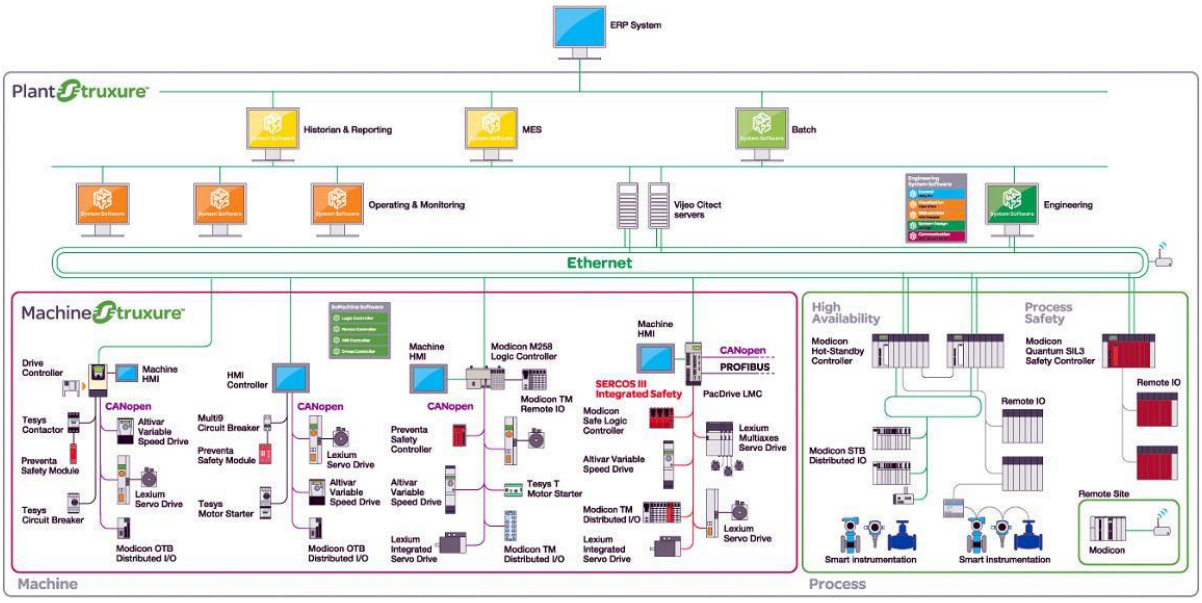
1.1 Mobilní aplikace v průmyslu

1.1.1 Průmyslové řídící systémy

Než se obrátím na popis mobilních aplikací pro průmyslové řídicí systémy, je nutné stručně se seznámit s moderními průmyslovými řídícími systémy. Jde o obecný termín, který kombinuje softwarové a hardwarové systémy používané v automatizačním průmyslu, včetně distribuovaných řídicích systémů (DCS), systémů řízení dispečinku a sběru dat (SCADA), programovatelných automatů (PLC), rozhraní člověk-stroj (HMI), systémů řízení výroby (MES) atd. [2].

V současné době jsou všechny továrny, obchodní centra, a dokonce i obytné budovy řízeny v jedné nebo druhé formě pomocí systémů řízení procesů. V souladu s obecnou definicí, systém automatizovaného řízení procesů shromažďuje data ze vzdálených stanic, zpracovává je a používá automatizované algoritmy nebo dispečerský program řízený operátorem k vytváření příkazů, které jsou pak odesílány na vzdálená zařízení (nebo „zařízení v terénu"). Poprvé se ICS objevila v letech 1970-1980 [2]. V posledním desetiletí však moderní technologie, včetně .XML, .Net, .JSON atd. začaly být stále více využívány v systémech řízení procesů. Samozřejmě, mobilní aplikace také nestály stranou.

Moderní infrastruktury pro automatizované systémy řízení procesů mají složitou architekturu skládající se z takových známých prvků jako jsou servery, počítače, síťové přepínače, softwarové technologie (.Net, DCOM, XML atd.) a složitější komponenty: programovatelné ovladače, vysílače, napájecí pohony, analogové řídící signály atd. Na obr. 1 je ukázáno schéma moderní infrastruktury systému řízení procesů. Věnujte pozornost třem hlavním úrovním.



Obr. 1: Architektura průmyslových řídicích systém [1]

1.1.2 Úrovně průmyslových řídicích systémů

1. Nižší úroveň, ve které se nachází většina akčních zařízení. Jak je uvedeno výše, jsou vhodné pro akční práci, například mohou: regulovat teplotu a tlak v reaktoru, provádět regulační operace, jako jsou otevíraní a uzavíraní ventilů, zapínání a vypínání čerpadel atd. Tato vrstva je sférou nízko úrovňových průmyslových protokolů, jako je Modbus, Modbus TCP, HART, Wireless HART, Profibus DP nebo PA, Foundation Fieldbus H1. Na této úrovni systému řízení procesů pracují inženýři nižší úrovně systému řízení procesů, elektrikáři, technici a další specialisté [2][1].

2. Střední úroveň, kde se můžete setkat s PLC na vysoké úrovni, distribuovanými řídicími systémy (DCS, systém řízení procesů, charakterizovaný budováním distribuovaného I/O systému a decentralizovaným zpracováním dat), systémy řízení dispečinku a systémy pro sběr dat (SCADA) softwarový balíček určený k vývoji nebo poskytování systémů pro shromažďování, zpracování, zobrazování a archivaci informací o monitorovacím nebo kontrolním objektu v reálném čase) a integraci rozhraní člověk-stroj (Human-Machine Interface (HMI)), stejně jako servery OPC (Open Platform Communications) [2][1]. Veškerá Inteligentní řízení probíhají zde. Operátoři a automatizované systémy na základě dat z nižších úrovní rozhodují a odesílají příkazy do přístrojů. Zde probíhá celý proces automatizace výroby. Operátoři, procesní inženýři, inženýři systémů řízení procesů, programátoři PLC pracují se systémy na této úrovni.

3. Vyšší úroveň integruje obchodní a průmyslové procesy. Tato vrstva poskytuje odkaz na podnikové sítě a systémy plánování podnikových zdrojů (ERP). Na této úrovni existují různé systémy řízení výrobních aktiv (PAS) a systémy řízení výrobních procesů (MES, které poskytují správné informace ve správný čas a ukazují pracovníkům s rozhodovací pravomocí na pracovišti, jak lze podmínky workshopů optimalizovat pro zvýšení produktivity) [2][1]. Manažeři a technický personál zde pracují s automatizovaným systémem řízení procesů.

1.2 Typy mobilních aplikací pro průmyslové řídicí systémy

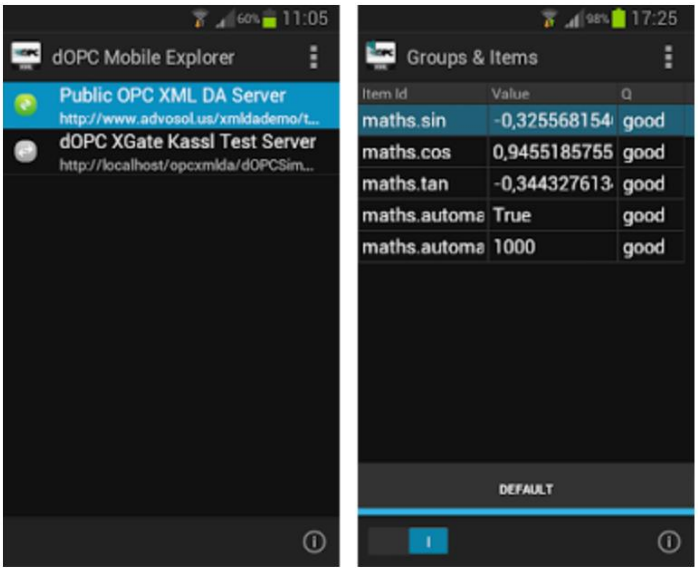
Jak už nyní vím, infrastruktura systému řízení procesů je obrovská. Mobilní aplikace pro systémy řízení procesů jsou také velké. Vzhledem k tomu, že klasifikace těchto řešení nebyla nalezena, bylo vyvinuto vlastní rozdělení, založené na vztahu aplikace k průmyslovému procesu a jeho umístění v architektuře průmyslového řídicího systému. Aplikace používané pro průmysl byly rozděleni na tři typy:

1. Ovládací panel: přímá konfigurace, monitorování, řízení výrobního procesu nebo jeho komponent. Všechny tyto aplikace mají jednu důležitou vlastnost: spojení mezi aplikací a průmyslovou složkou se vyskytuje v údajně bezpečném prostředí, někde na nižší a střední úrovni systému řízení procesů.

- Konfigurace PLC. Nastavení nebo řízení komponent systému řízení procesů: PLC, HMI, připojené k PLC, RTU a dalším. Mobilní aplikace se připojuje ke komponentě systému řízení průmyslových procesů. GoProbe app [3].
- Mobilní panel HMI. Panel HMI uvnitř mobilního zařízení pro ovládání několika průmyslových komponent. Umožňuje konstruktérovi konfigurovat panel HMI a připojit jeho komponenty k PLC nebo jiným zařízením přes Modbus/TCP, OPC nebo jiné protokoly a rozhraní. Příklad: aplikace robota společností robomotion GmbH (znázorněna na obr.2) a SOTEC GmbH, ScadaTouch Basic, HMI-Modbus [3].

Obr. 2: aplikace robotu společností robomotion GmbH [3]

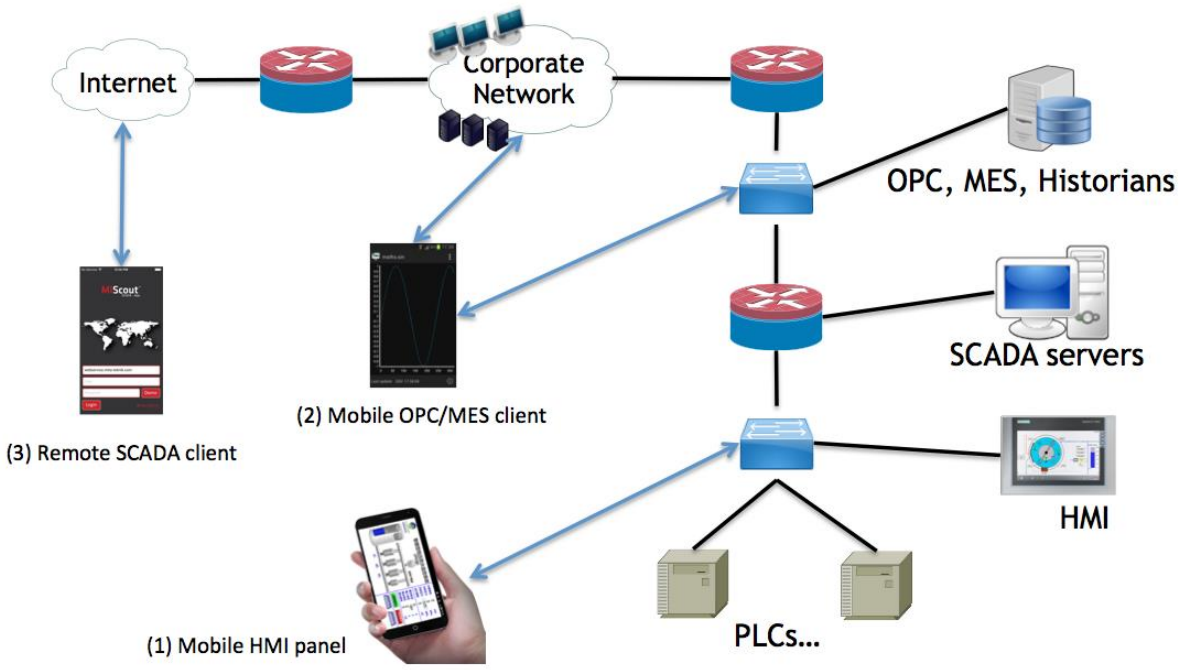
2. Klient pro OPC / MES nebo archivační systém: archivační aplikace umožňují inženýrovi nebo vlastníkovi procesu číst a interpretovat některé informace ze střední a nejvyšší úrovně komponent systému řízení procesů. Data jsou jen pro čtení a uživatel nemá ani přímý přístup k PLC, HMI nebo SCADA – pouze schopnost zobrazit určité proměnné. Do této kategorie jsou přiřazeni mobilní klienti pro aplikace MES. Obvykle se používají ke čtení dat z archivního serveru nebo agregovaných dat ze systému MES. Typickou aplikací z této kategorie je klient (prohlížeč) pro OPC. Není připojen přímo ke komponentám systému řízení procesů, ale k OPC serveru. K propojení obvykle dochází na nejvyšších úrovních infrastruktury. Kromě toho může být spojení provedeno na dálku. Příklad použití aplikace OPC XML DA Explorer (znázorněna na obr.3) [5].



Obr. 3: aplikace OPC XML DA Explorer [5]

3. Dálkové ovládání SCADA: aplikace, které umožňují vzdáleně (mimo výrobní síť) monitorovat / řídit výrobní proces. Přestože řešení první kategorie jsou pro to vhodná, vývojáři tuto příležitost nepodporují ani o ní neinformují. Požadavky na bezpečnost a spolehlivost těchto aplikací jsou nejpřísnější. Příklad: Pro-face Remote HMI (znázorněna na obr.4) [4]. Na obr. 5 je uvedeno v jaké části architektury se nachází různé typy aplikací.

Obr. 4: aplikace Pro-face Remote HMI [4]



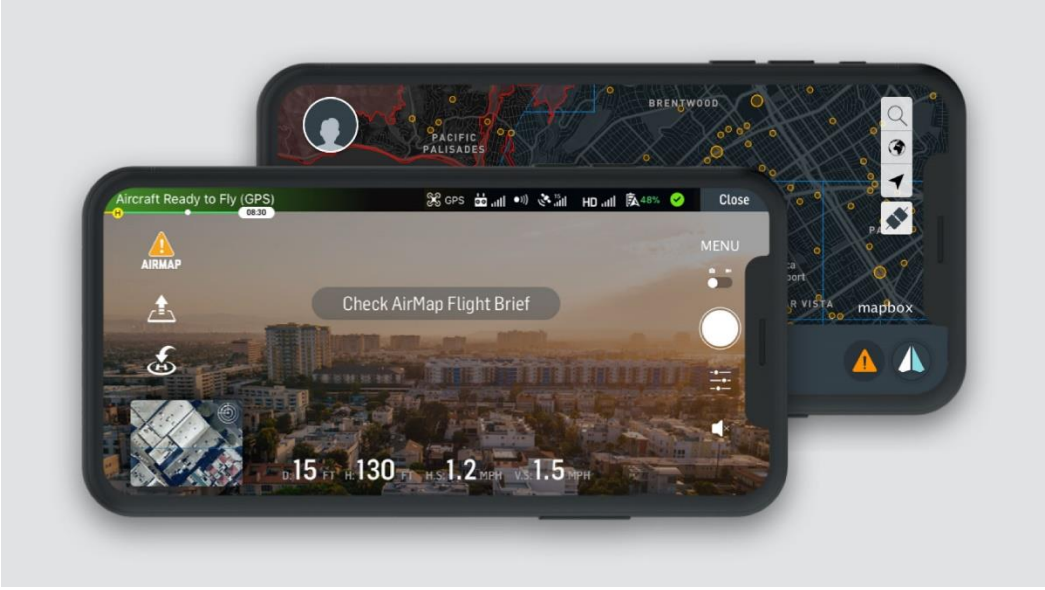
Obr. 5: Aplikace v architektuře průmyslových řídicích systémů

1.3 Mobilní aplikace v hobby sektoru

V dnešní době se mobilní aplikace pro řízení velmi často používají v hobby sektoru. To je zapříčiněno kvůli rozvoji dronů na občanském trhu. Výrobci dronů usilují o vytvoření realistického pocitu letu s použitím různých aplikací a kamer první osoby, pro zlepšení nebo nahrazení standardní fyzické kontroly.

Obr.6 aplikace od Ar.Drone [6]

Rozhraní člověk-stroj v takových aplikacích připomíná hry což můžeme vidět na obr. 6, ale existují taky i více složitější programy připomínající ve své struktuře průmyslové analogy což je zobrazeno na obr. 7.



Obr. 7: DJI discovery [7]

V případě analýzy mobilních aplikací pro hobby sektor lze lehce vidět jejich podobnost s aplikacemi typu ovládací panely z jejich průmyslových analogů. Většina rozlišení takových programů pro hobby sektor a průmysl spočívá v komunikačních protokolech používaných v těchto zařízeních.

1.4 Mobilní aplikace v průmyslu a hobby sektoru ohledně jejich návrhů

a) Ovládací panel

Při návrhu aplikací takového typu musíme dát pozor na chybějící ověřovaní dat na straně serveru z pohledu průmyslového procesu. Vzhledem k tomu, že serverová aplikace zcela důvěřuje údajům, které pocházejí od operátora a správnost technologického procesu zcela závisí na akcích a příkazech které odesílá, mohou jakékoliv nedostatky ve validaci dat vést k fatálním následkům. Současně je v mobilní aplikaci možné provést chybu při spatné volbě čísla nebo neúmyslně nastavit zápornou hodnotu parametru. Nesprávná interpretace hodnot může také vést k odmítnutí služby pro serverové i klientské aplikace. Tyto podmínky se netýkají mobilních aplikací pro hobby sektor.

b) OPC / MES klient nebo archivační systém a Dálkové ovládání

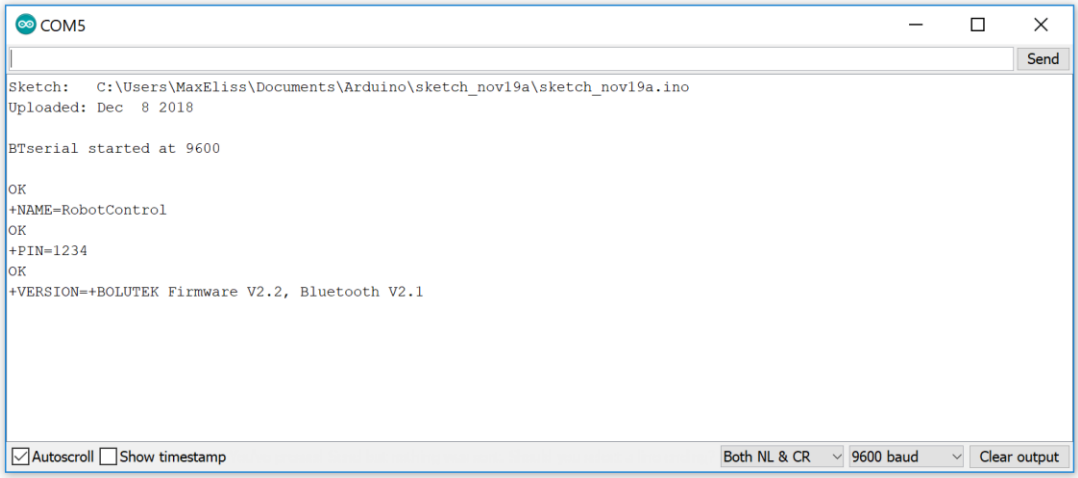
Vzhledem k tomu, že tyto aplikace již nejsou v chráněném systému a jsou přenášeny přes sítě pro všeobecné účely (Internet), server a aplikace musí být zabezpečeny šifrováním dat, která přecházejí z aplikace na server. To znamená, že k systému řízení musí být také přiřazen zabezpečený autorizační systém.

2 Mobilní aplikace pro řízeni laboratorního robota

2.1 Platforma realizace

Na začátku této kapitoly bude vysvětleno, proč jako platforma pro program byla vybraná mobilní platforma Android. Poprvé mnou byla vybrána mobilní platforma, která umožňuje používat robota kdykoliv a bez doplňujícího zařízení. V dnešní době na trhu máme dvě mezi sebou konkurující mobilní platformy IOS a Android.

1. Prvním a hlavním důvodem výběru Android pro mou práci je ten, že jeho Hardware podporuje všechny verze Bluetooth. Robot se kterým jsem pracoval má bluetooth verze 2.1 a konkurent Androidu v dnešní době podporuje Bluetooth od verze 4.0 (obr.8) [8].



Obr. 8: Verze použitého modulu

2. Druhým důvodem byl ten, že Android používá ve světě mnohem více uživatelů než jeho nejbližší konkurent, což je uvedeno na obr. 9 [10][11].

Period	Android	iOS	Windows Phone	Others
2016Q1	83.4%	15.4%	0.8%	0.4%
2016Q2	87.6%	11.7%	0.4%	0.3%
2016Q3	86.8%	12.5%	0.3%	0.4%
2016Q4	81.4%	18.2%	0.2%	0.2%
2017Q1	85.0%	14.7%	0.1%	0.1%

Source: IDC, May 2017

Obr. 9: porovnaní popularity mobilních platform [10]

3. Jako třetí důvod bych uvedl to, že je platforma Android pro programátora celkově více přátelská než u konkurenta. Například u IOS možnost kompilace programu možná jenom v MAC operation systém [8].

2.2 Jazyk programovaní

V této části mnou budou prozkoumávány programovací jazyky a bude vybrán ten nejvhodnější pro moji úlohu. Celkově jsou 2 oficiální jazyky, které Google podporuje, ale možné je využít skoro všechny existující [12][9]. Jediným důvodem, proč používat jiné jazyky než ty, které podporuje Google, jsou specifické úlohy pro mobilní platformy jako zpracování a renderování grafiky, videa a komplexních 3D modelů, což je v dnešní době pro mobilní platformy málo frekventovaná oblast.

a) Java - plusy a mínusy

Obr. 10: Java [9]

Java je oficiální programovací jazyk podporovaný vývojovým prostředím Android Studio. Podle ročního průzkumu zdroje Stackoverflow, byla Java v roce 2018 jedním z pěti nejpopulárnějších programovacích jazyků. Prvním krokem k zvládnutí jazyka Java je instalace aplikace Android Studio. Jedná se o typ softwaru s názvem IDE – Integrated Development Environment nebo vestavěné vývojové prostředí. Android Studio je dodáván s Android SDK – sada nástrojů pro vývoj Android a vše, co je potřeba pro spolehlivý start [11].

Při mé práci bylo zjištěno, že Java je odkazována na většinu oficiálních dokumentů Google a hledání placených a bezplatných knihoven a příruček je snadné a existuje jich poměrně hodně. Většina aplikací pro Android je napsána v jazyce Java a vlastní jazyk Java umožňuje vývojářům nejen vytvářet nové aplikace, ale také udržovat stávající a pracovat s existujícím zdrojovým kódem. Bohužel, složitost Javy ztěžuje každé programování. Stejně jako objektově orientovaný programovací jazyk, má spoustu funkcí ve formě třídních konstruktérů, výjimek které vedou k pádu aplikací během provozu a další body, které musí být při vývoji vždy zohledněny. Kód Java je však snadno čitelný a strukturovaný, zejména pokud jsou respektovány přijaté standardy jeho návrhu.

Při vývoji v jazyce Java pro Android se používají nejen třídy jazyka Java, které obsahují kód, ale také soubory manifestu XML, které poskytují systému základní informace o programu a automatických sestavovacích systémech Gradle, Maven nebo Ant, jejichž příkazy jsou napsány v Groovy, POM a XML; Gradle je standardně používán v projektech a v počátečních fázích učení, pro vývoj v jazyce Java, nebude třeba soubory napsané v Groovy upravovat. XML je také běžně používán pro rozložení uživatelského rozhraní. Android Studio uznávané společností Google, jako oficiální vývojové prostředí od prosinci 2014 pro operační systém Android, se

každým rokem zlepšuje, což usnadňuje život vývojářům Android. Jeho funkce, jako je vizuální editor UI a dokončení kódu, usnadňují vývojový proces [12].

b) Kotlin jako nadstavba na Javu

Obr. 11: Kotlin [9]

Jazyk byl oficiálně představen v květnu 2017 na Google I / O a je umístěn na Google jako druhý oficiální programovací jazyk Android po Javě, jen je o něco jednodušší k pochopení [12]. Znalost jazyka Java je zde potřebná k pochopení principů Kotlin, obecné struktury jazyka a jeho vlastností. Mnoho vývojářů pokládá Kotlin za obálku kolem Javy a doporučuje jej naučit se až poté, co se budete cítit sebejistě ve svých znalostech jazyka Java. Kotlin je kompatibilní s jazykem Java a nezpůsobuje snížení výkonu ani zvýšení velikosti souboru. Rozdíl oproti Javě spočívá v tom, že vyžaduje méně služeb, tzv. Kotevní kód, čímž je jednodušší a přehlednější. Jeho tvůrcům se podařilo vyhnout se nullpointexceptions, a kompilace již není přerušena kvůli maličkostem jako je zapomenutý znak „;“ [12].

c) C/C++ používání nízko úrovňových jazyků

Obr. 12: C/C++ [9]

Jazyky nižší úrovně, které jsou podporovány aplikací Android Studio pomocí jazyka Java NDK. To umožní psát nativní aplikace, které mohou být užitečné pro vytváření her nebo jiných programů náročných na zdroje. Android Studio nabízí podporu C / C ++ přes Android NDK (Native Development Kit). To znamená, že kód nebude probíhat

přes Java Virtual Machine, ale přímo přes zařízení, což vám poskytne větší kontrolu nad takovými prvky systému, jako je paměť, senzory, gesta atd. To také znamená, že budete muset používat knihovny napsané v jazyce C nebo C++ [12][9]. Na druhou stranu je složité a ne příliš pohodlné konfigurovat, takže se doporučuje používat jej pouze pro zápis těch modulů programu, kde potřebujete rychle provádět komplexní operace: zpracování a vykreslování grafiky, videa a složitých 3D modelů.

Závěr: Jako hlavní programovací jazyk byla vybráná Java, kvůli jejímu největšímu obsahu hotových knihoven pro práci s bluetooth, což velmi urychlí a usnadní napsání kódu a oproti tomu, bude aplikace určena pro posílání a zpracování jednoduchých signálů na řízení a nebude potřebovat tak velký výkon. Takže alespoň jedna třída bude napsána v jazyce Kotlin pro zkoušení nového moderního jazyka a kvůli tomu, že je plně kompatibilní s Javou nebude překážet už napsaným třídám v Javě.

2.3 Modelování struktury programu, napsání kódu a základy práce s Android studio

V této části práce bude nejen popis struktury programu, ale také vysvětlena specifita programování pro Android. Zaprvé zde musím vysvětlit základy Android programovaní. Hlavně to, jak fungují Aktivity, Fragment apod.

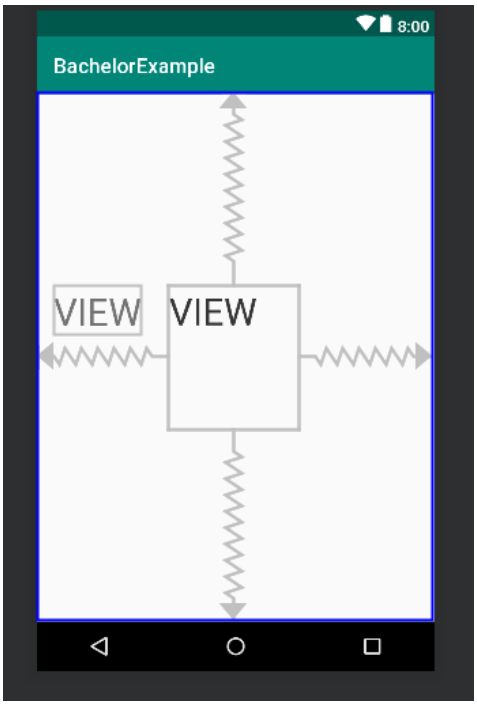
2.3.1 Základy programovaní pro Android

Komponenty aplikací jsou stavebními kameny aplikace Android. Každá komponenta je samostatný bod, přes který může systém vstoupit do aplikace. Ne všechny komponenty jsou body vstupu uživatele, a některé z nich na sobě závisí. Každá komponenta je navíc samostatnou konstrukční jednotkou a hraje určitou roli – každá z nich představuje jedinečný prvek struktury, která určuje fungování aplikace jako celku. Aplikační komponenty lze přiřadit jednomu ze čtyř typů. Komponenty každého typu jsou určeny pro konkrétní účel a mají svůj vlastní životní cyklus, který určuje způsob, jakým je komponenta vytvořena a ukončena.

a) Activity a jeho životní cyklus

Pokud vymyslím analogii s Windows, aplikace se bude skládat z oken nazvaných Activity. V určitém čase se obvykle zobrazuje jedna aktivita a zabírá celou obrazovku, a aplikace se mezi nimi přepíná. Zvážím například poštovní aplikaci. V ní je jedna aktivita seznam písmen, další je pohled na dopisy a třetí je nastavení poštovní schránky. Při práci se s nimi pohybuju [9].

Obsah aktivity je tvořen z různých komponent s názvem View. Nejběžnějším tlačítkem je tlačítko, vstupní pole a zaškrtávací políčko atd. Na obr. 13 je vidět příklad Activity.



Obr. 13: Activity zobrazena editorem

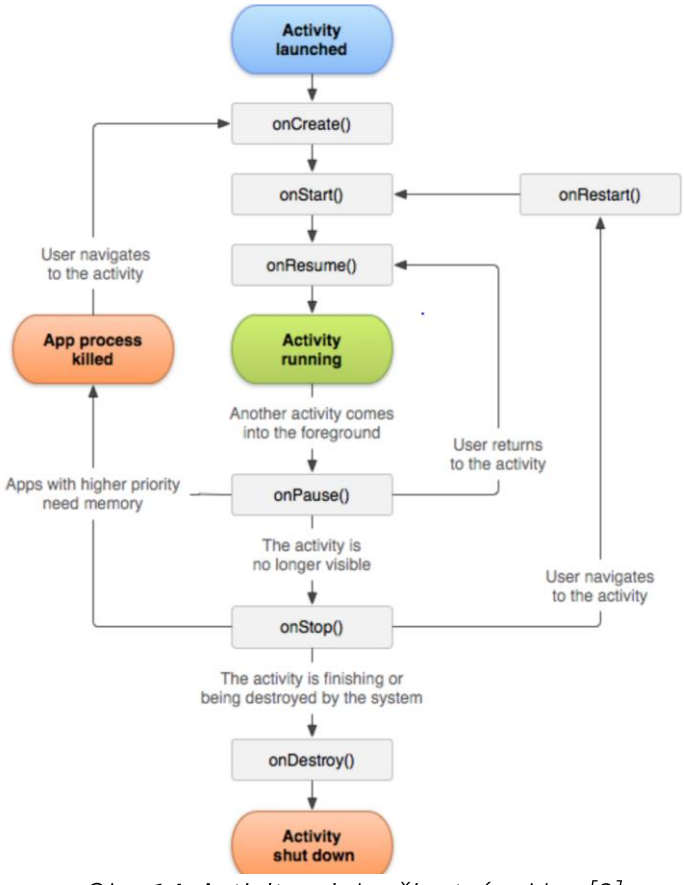
Když je aplikace spuštěna, vytvořím novou activitu a zavřu starou, minimalizuju aplikaci, znovu ji otevřu atd. Činnost může zvládnout všechny tyto pohyby. To je nutné například pro uvolnění zdrojů nebo uložení dat. V nápovědě je popsána dostatečně podrobně.

Aktivita vytvořená během provozu aplikace může být v jednom ze tří stavů:

- Resumed - Aktivita je viditelná na obrazovce, je zaostřena a uživatel s ní může pracovat. Tento stav se také někdy nazývá Běh [9].

- **Paused** - Aktivita není zaostřena, uživatel s ní nemůže pracovat, ale je viditelná (je blokována jinou aktivitou, která nezabírá celou obrazovku nebo je průsvitná) [9].
- **Stopped** - Aktivita není viditelná (zcela překrývající se jinou aktivitou), resp. není zaostřena a uživatel s ní nemůže komunikovat. [9]

Když aktivita jede z jednoho stavu do druhého, systém volá své různé metody, které můžu vyplnit vlastním kódem. Schematicky to může být reprezentováno jako na obr. 14.



Obr. 14: Activity a jeho životní cyklus [9]

Mám tedy následující metody aktivity, které systém volá (obr.14):

- **onCreate ()** - volá při prvním vytvoření aktivity;
- **onStart ()** - volá před činností, bude viditelný pro uživatele;
- **onResume ()** - volá před tím, než je k dispozici pro činnost uživatele (interakce);
- **onPause ()** - volá před zobrazením jiné aktivity;

- onStop () - volá, když aktivita již není uživateli viditelná;
- onDestroy () - volá před zničením aktivity.

Tyto metody nezpůsobují změnu stavu. Naopak, změna stavu aktivity je spouštěč, který tyto metody volá. Na tuto změnu si musím dát pozor, a musím na ni odpovídajícím způsobem reagovat.

Příklad:

```
package cz.cvut.fs.robduino3;

import android.app.Activity;
import android.content.Intent;
import android.os.Bundle;
import android.view.View;

public class ConnectActivity extends Activity {


    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.menu_connect);
    }

    public void gonext2(View view) {
        Intent intent
= new Intent(this, ChooseTypeConnectActivity.class);
        startActivity(intent);
    }

    public void back(View view) {
        Intent intent = new Intent(this, MainActivity.class);
        startActivity(intent);
    }
}
```

b) Service a jeho použití

Service je komponenta aplikace, která může provádět dlouhodobé operace na pozadí a neobsahuje uživatelské rozhraní. Další komponenta aplikace může spustit službu, která bude i nadále pracovat na pozadí i když uživatel přepne na jinou aplikaci. Navíc se komponenta může vázat na službu, aby s ní mohla komunikovat, a dokonce provádět interprocesovou komunikaci (IPC). Služba může například

zpracovávat transakce v síti, přehrávat hudbu, provádět I / O souborů nebo komunikovat s poskytovatelem obsahu, a to vše se děje na pozadí [9].

Služba může mít dvě podoby:

Spuštěna:

Služba je spuštěna, když ji komponenta aplikace (například operace) spustí voláním startService () [9]. Po spuštění může služba běžet na pozadí po neomezenou dobu, i když je komponenta která ji spustila zničena. Spuštěná služba obvykle provádí jednu operaci a nevrací výsledky volající komponentě. Může například stáhnout nebo nahrát soubor přes síť. Po dokončení operace by se služba měla zastavit sama.

Přivázaná:

Služba je „vázána“, když je na ni komponenta aplikace vázána voláním bindService (). Přidružená služba nabízí rozhraní klient-server, která umožňuje komponentám komunikovat se službou, odesílat požadavky, získávat výsledky, a dokonce to provádět mezi různými procesy prostřednictvím procesové komunikace (IPC) [9]. Připojená služba funguje pouze pokud je k ní připojena další součást aplikace. Současně může být ke službě připojeno několik komponent, ale když je vše odpojené, služba je zničena.

Chcete-li vytvořit službu, musíte vytvořit podtřídu třídy služby (nebo jednu z jejich existujících podtříd). Ve vaší implementaci musíte přepsat některé metody zpětného volání, které zpracovávají klíčové body v životním cyklu služby, a v případě potřeby poskytnout mechanismus pro přiřazení komponent.

Příklad použití třídy Service:

```
public class HelloIntentService extends IntentService {


  public HelloIntentService() {
      super("HelloIntentService");
  }
  @Override
  protected void onHandleIntent(Intent intent) {
      long endTime = System.currentTimeMillis() + 5*1000;
      while (System.currentTimeMillis() < endTime) {
          synchronized (this) {
              try {
                  wait(endTime - System.currentTimeMillis());
              } catch (Exception e) {
              }
```

```
            }
        }
    }
}
```

V mém programu, tedy pro laboratorního robota není nutno používat Service, protože většinou neobsahuje aplikace, které by musely běžet bez použití programu.

c) Content provider a jeho použití

Content provider (poskytovatel obsahu) spravuje celkovou sadu dat aplikace. Data mohou být uložena v souborovém systému, databázi SQLite, na internetu nebo jiném trvalém úložišti, ke kterému má vaše aplikace přístup. Prostřednictvím poskytovatele obsahu mohou jiné aplikace požadovat nebo dokonce data modifikovat (pokud to poskytovatel obsahu povoluje). Například v systému Android existuje poskytovatel obsahu, který spravuje kontaktní informace uživatele. Každá aplikace, která obdržela příslušná oprávnění, může požádat část tohoto poskytovatele obsahu (například ContactsContract.Data) o čtení a zápis informací o konkrétní osobě [9].

Poskytovatel obsahu poskytuje data externím aplikacím ve formě jedné nebo více tabulek, podobných tabulkám v relační databázi. Řádek je instancí určitého typu dat shromážděných poskytovatelem a každý sloupec v tomto řádku je samostatná položka dat shromážděných pro instanci.

V mém případě jedním dodavatelem kontentu bude okolní prostředí, ze kterého budu sbírat data abych z ní našel konkrétně bluetooth modul robota, se kterým musí být vytvořena komunikace.

Příklad sbíraní kontentu:

```
    //Turn on searching Bluetooth devices
    private void searchDevice() {
        Log.d(TAG, "searchDevice()");
        enableBluetooth();
        //must control Android version and give geo location(must to
be ,the rule from google)
        checkPermissionLoc();
        IntentFilter filter
= new IntentFilter(BluetoothAdapter.ACTION_DISCOVERY_STARTED);
```

```
        filter.addAction(BluetoothAdapter.ACTION_DISCOVERY_FINISHED);
        filter.addAction(BluetoothDevice.ACTION_FOUND);
        registerReceiver(myReceiver, filter);

        if (!myBluetoothAdapter.isDiscovering()) {
            Log.d(TAG, "searchDevice :Start searching devices");
            myBluetoothAdapter.startDiscovery();

        }
        if (myBluetoothAdapter.isDiscovering()) {
            Log.d(TAG, "searchDevice :searching already
running...restarting");
            myBluetoothAdapter.cancelDiscovery();
            myBluetoothAdapter.startDiscovery();

        }
    }
```

d) Broadcast receiver a jeho použití

Přijímač vysílání (Broadcast Receiver) je komponenta, která reaguje na oznámení distribuovaná v celém systému [9]. Mnohá z těchto oznámení jsou zasílána systémem – například oznámení o vypnutí obrazovky, vybití baterie nebo pořízení fotografie. Reklamy mohou být také odesílány aplikacemi – například pro informování jiných aplikací, že některá data byla stažena do zařízení a je nyní připravena k použití. Přestože rozhlasové přijímače nemají uživatelské rozhraní, mohou ve stavovém řádku vytvářet oznámení, která upozorní uživatele na událost „reklamní vysílání". Nejčastěji jsou však jen „bránou" pro ostatní komponenty a jsou navrženy tak, aby vykonávaly minimální množství práce. Mohou například spouštět službu pro provádění určitých akcí, když nastane událost. Vysílací přijímač patří do podtřídy třídy BroadcastReceiver, a každá taková zpráva je poskytována jako objekt Intent. Komponenta BroadcastReceiver bude běžně použita v různých částech programu, ale pro můj případ bude většinou sloužit jako komunikace s uživatelem.

Příklad:

```
    //    Monitoring bluetooth status
//    On / Off and search for new devices
    private BroadcastReceiver myReceiver = new BroadcastReceiver() {
        @Override
        public void onReceive(Context context, Intent intent) {
            final String action = intent.getAction();
```

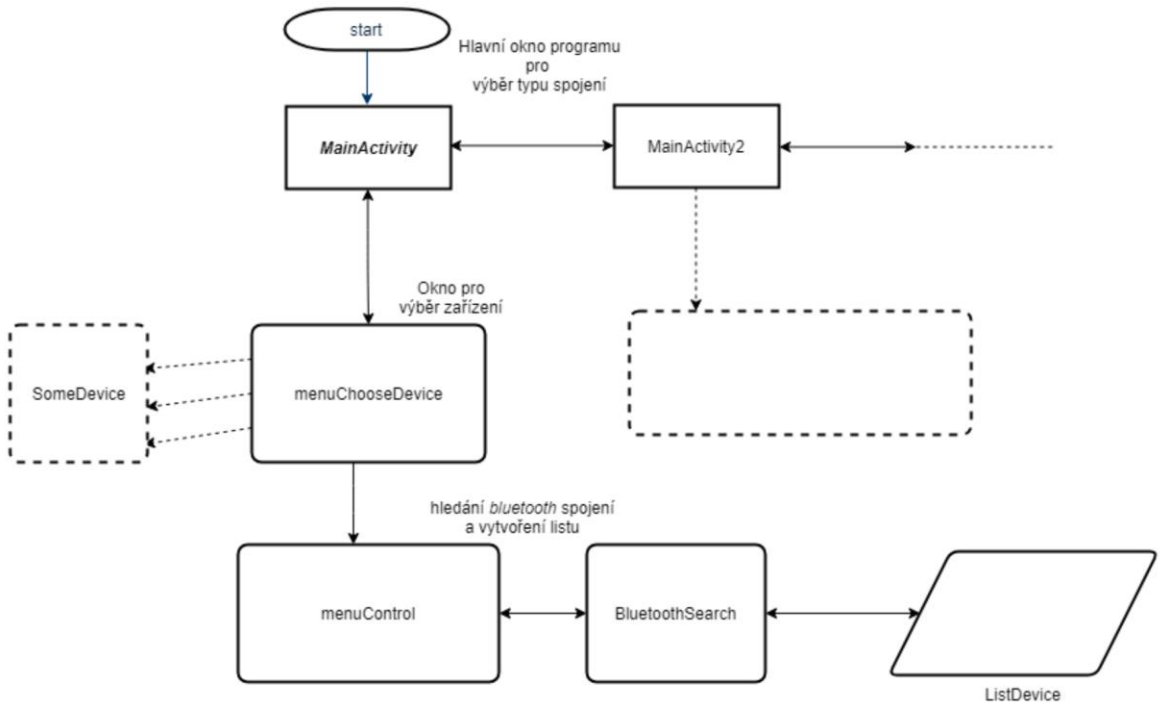
```
            //start new searching
            if (action.equals(BluetoothAdapter.ACTION_DISCOVERY_STARTE
D)) {
                Log.d(TAG, "onReceiver : ACTION_DISCOVERY_STARTED");
                showMessage("Start searching devices");
                myProgressBar
= ProgressDialog.show(ControlActivity.this, "Searching
Devices", "Please wait");
            }
            //search ended
            if (action.equals((BluetoothAdapter.ACTION_DISCOVERY_FINIS
HED))) {
                Log.d(TAG, "onReceiver : ACTION_DISCOVERY_FINISHED");
                myProgressBar.dismiss();
                showMessage("Search end");
                showListDevices();
            }
            //if searched new device
            if (action.equals(BluetoothDevice.ACTION_FOUND)) {
                Log.d(TAG, "onReceiver : ACTION_FOUND");
                BluetoothDevice device =
intent.getParcelableExtra(BluetoothDevice.EXTRA_DEVICE);
                if (device != null) {
                    if (!myDevices.contains(device)) {
                        myDeviceList.add(device);
                    }

                }

            }
        }
    };
```

V této části kódu provádí zapisování vyhledaných bluetooth modulů do listu a ukazování různých zpráv pro uživatele, např. když se našel nějaký modul, musí se ukázat tento modul, jeho název a Mac adresa nebo když se hledaní skončilo, napíše zprávu, že je hledaní ukončeno.

2.3.2 Návrh celkové struktury programu

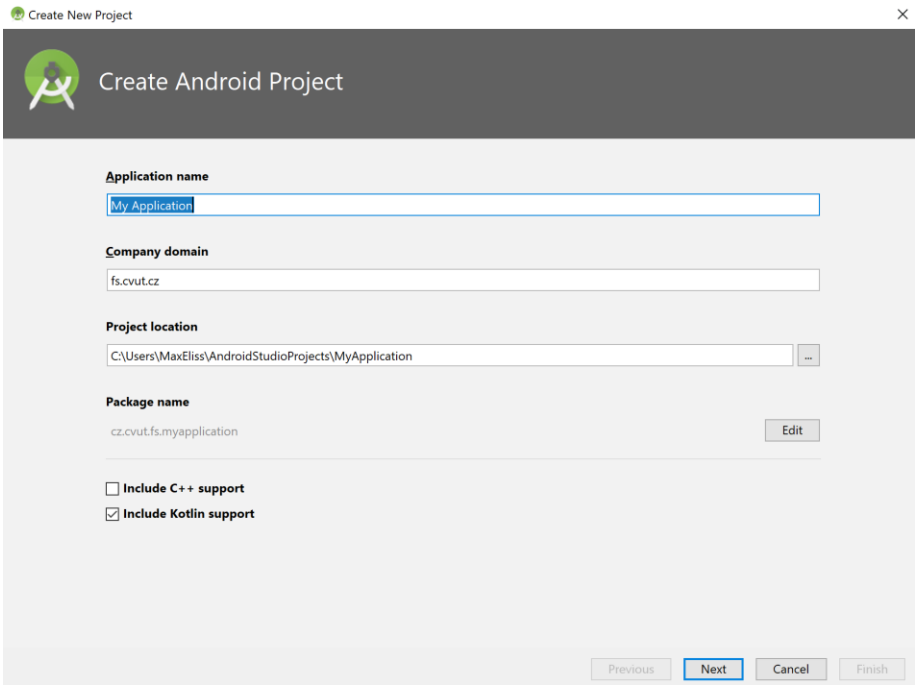
V této kapitole bude probíhat navrhováni programu a psaní kódu. Dál bude vysvětlován princip programovaní pro Android a během návrhu bude vysvětleno, jak pracovat v Android Studio.

Nejprve vytvořím jednoduchý návrh, který bude většinou řízen pomoci GUI a intuitivně jasných obrázků se kterými bude pracovat uživatel. Na obr. 15 je ukázáno zjednodušené schéma pro celkovou aplikaci, sama o sobě na začátku představuje minimalizované menu, ve kterém vybírám jednoduché nastavení - například typ spojení apod. Dále přecházím na tu nejdůležitější a nejsložitější část, jako je menu řízení, ve kterém prohází spojovaní, hledání a vytvoření seznamu s daty. Celkově struktura aplikace připomíná strom a pro přidáni například dalšího akčního prvku nebo jiného typu spojení, je potřeba vytvořit další větev v MainActivity nebo v menuChooseDevice, která může být zcela nezávislá na předchozím kódu. Například když potřebuju vytvořit další větev s novým motorem mám dvě možnosti, první okopírovat část kódu pro připojování k Bluetooth modulu a proudu dat z Bluetooth modulu do nové větvě. Druhý způsob je o něco těžší, ale umožňuje ušetřit paměť, a je možné použít tak známou reflexi, která ruší jednu z paradigmat OOP známou jako zapouzdření, a dává přístup k metodám typu private a protected. Jak funguje reflexe a příklad použití v mém programu je uveden v kapitole 2.3.4.



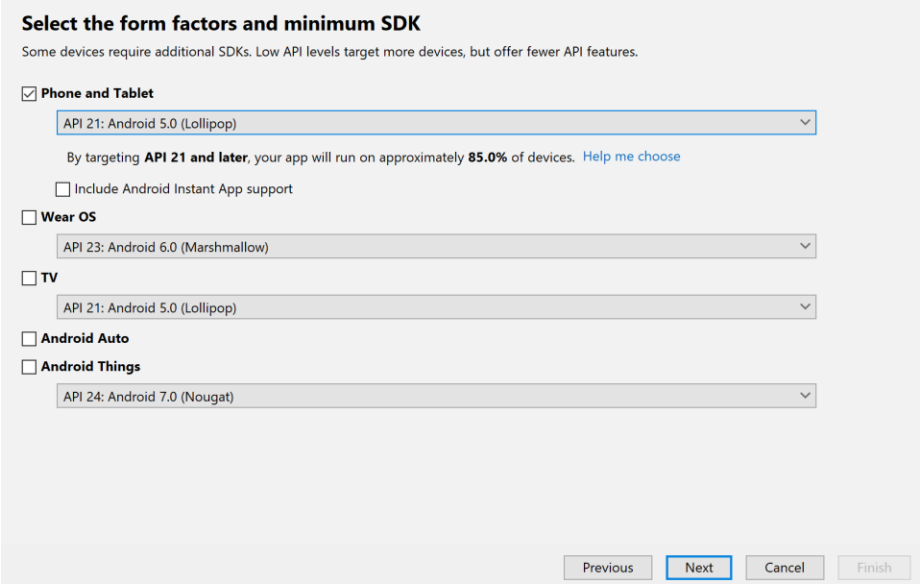
Obr. 15: Struktura programu

První krok vytvořeni projektu v Android studio a úprava xml souboru. Otevřu Android Studio a v prvním okně se zobrazí jméno aplikace Company domain, která slouží jako package a bude potřebná, když aplikace bude na PlayMarket, dále můžu zapnout podporu jiných jazyků (obr.16).



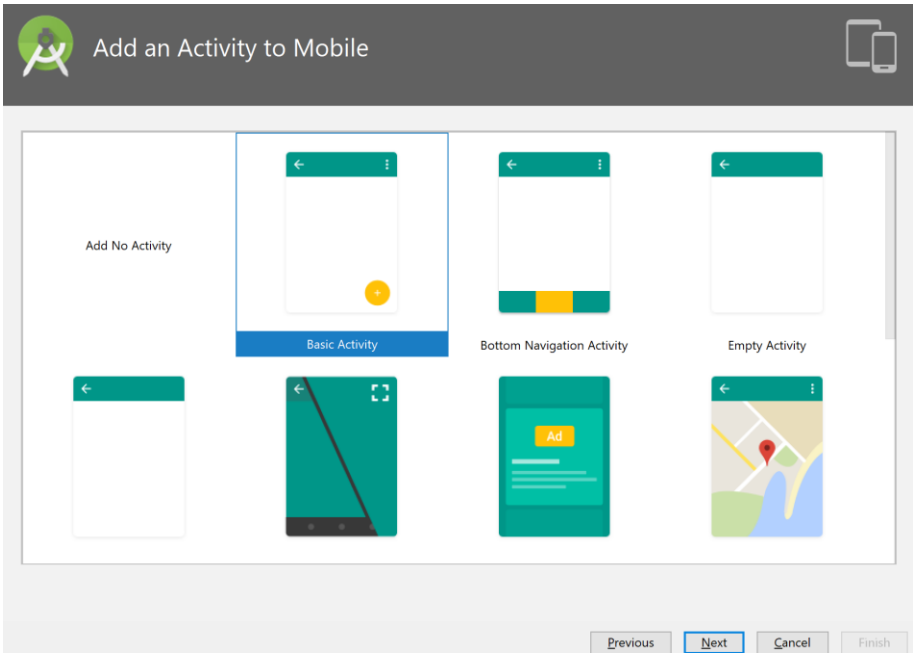
Obr. 16: Android studio vytvořeni projektu 1

V dalším okně vznikne výběr, na jaký typ zařízení budu dělat program, a jaký API (Application Programming Interface) budu používat, jak lze vidět na obr.17 různé verze Android podporuje různé API. Čím starší API použiji, tím méně funkcí budu mít, ale větší počet podporovaného zařízení.



Obr. 17: Android studio vyběr API

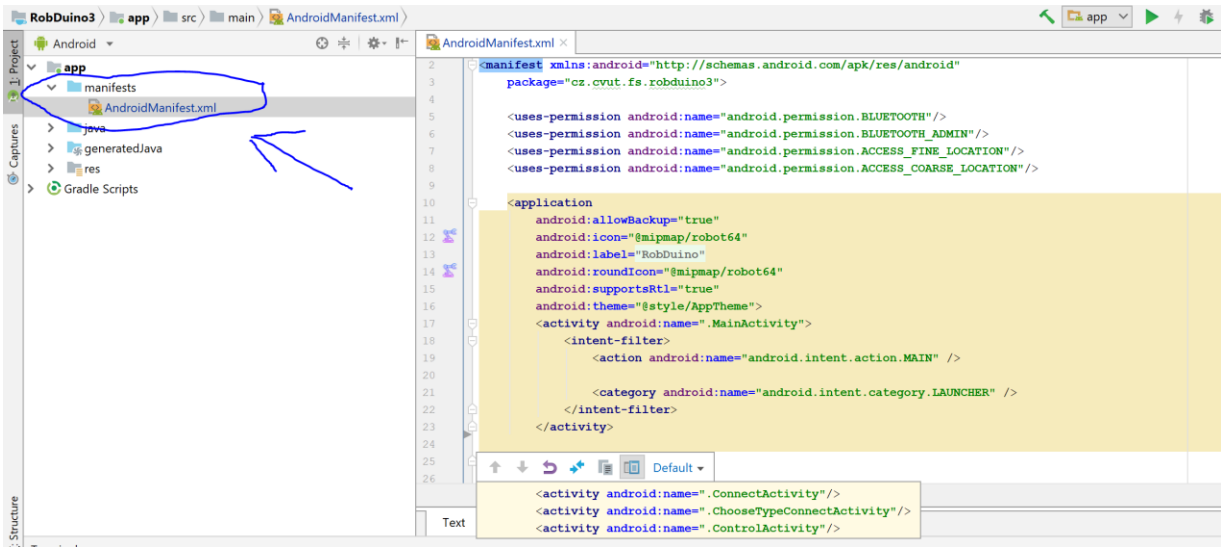
Dále nám Android nabídne automaticky vytvořit jednu Activity, pro svoji úlohu jsem vzal prázdné Activity které jsem pak upravil a dále pojmenoval Activity, a zmáčkl tlačítko finish.



Obr. 18: Android studio počáteční Activity

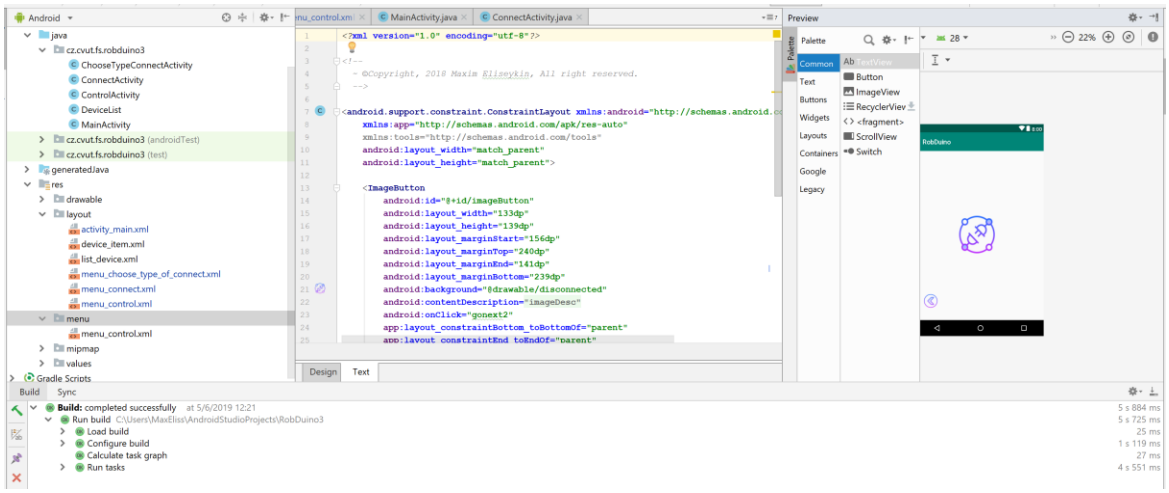
Za prvé musím ve složce manifest, která je ukázána on obr.18, upravit nebo vytvořit soubor AndroidManifest.xml, ve kterém musím ukázat všechny třídy aplikací

a ukázat Main třídu, takže musím povolit přístup aplikace k bluetooth modulu na mobilu (permission.BLUETOOTH,BLUETOOTH_ADMIN). Ještě je na obrázku 19 vidět speciální povolení (ACCESS_FINE_LOCATION, ACCESS_COARSE_LOCATION) pro přístup k přesné a přibližné poloze, bez které celá aplikace nebude fungovat, je to způsobeno tím že Google musí vědět do jaké lokální části PlayMarket tuto aplikaci přiradit (obr.19).



Obr. 19: Manifest soubor

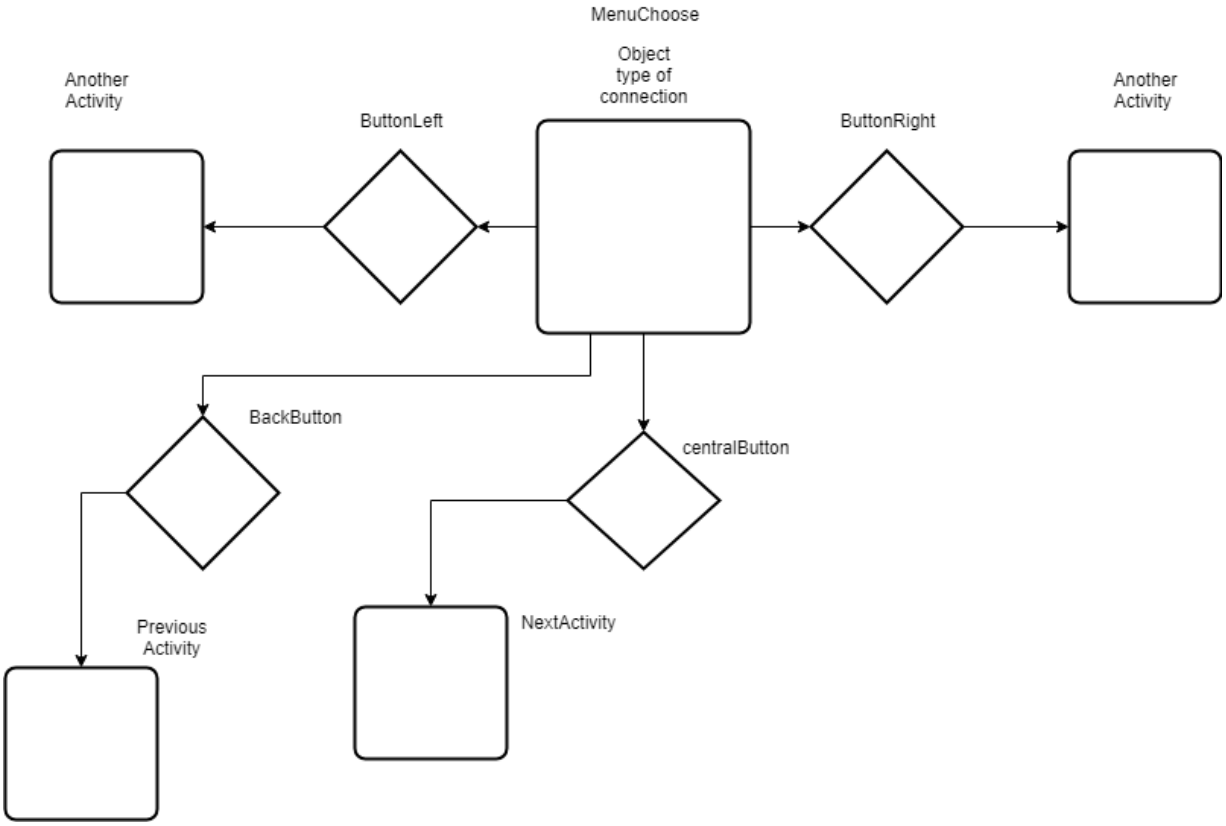
Dále skočím do složky res, vyberu složku mipmap a dodám do ní potřebné obrázky pro všichna tlačítka. Poté ve složce layout vytvořím všechny xml soubory, které budu potřebovat v mojí aplikaci. A .xml soubory můžu vytvořit pomocí WYSIWYG editoru, který už je v Android Studio (obr.20).



Obr. 20: WYSIWYG editor

2.3.3 Struktura tříd pro pohyb mezi různými Activity

Nyní začnu vytvářet třídy pro MainActivity a menuChooseDevice, které představují jednouchou strukturu, která je uvedena na obr.21. Jediným rozdílem mezi nimi je v tom, že MainActivity neobsahuje tlačítko zpět.



Obr. 21: Struktura tříd pro pohyb mezi activity

Realizace v kódu je celkem jednoduchá. Vytvořím tři zcela podobné metody, dvě z nich metoda Back (View v) a nextActivity4 (View v), vystupuje jako Akce k tlačítkem pomoci knihovny content a třídy intent, ve které jako nový objekt bude třída, kterou budu volat při akci. Metoda onCreate() slouží v Androidu jako základ, který volá XML soubor odpovídající za konkrétní grafickou část aplikace a vždycky musí být použita při práci GUI.

```
import android.app.Activity;
import android.app.AlertDialog;
import android.app.Activity;
import android.app.AlertDialog;
```

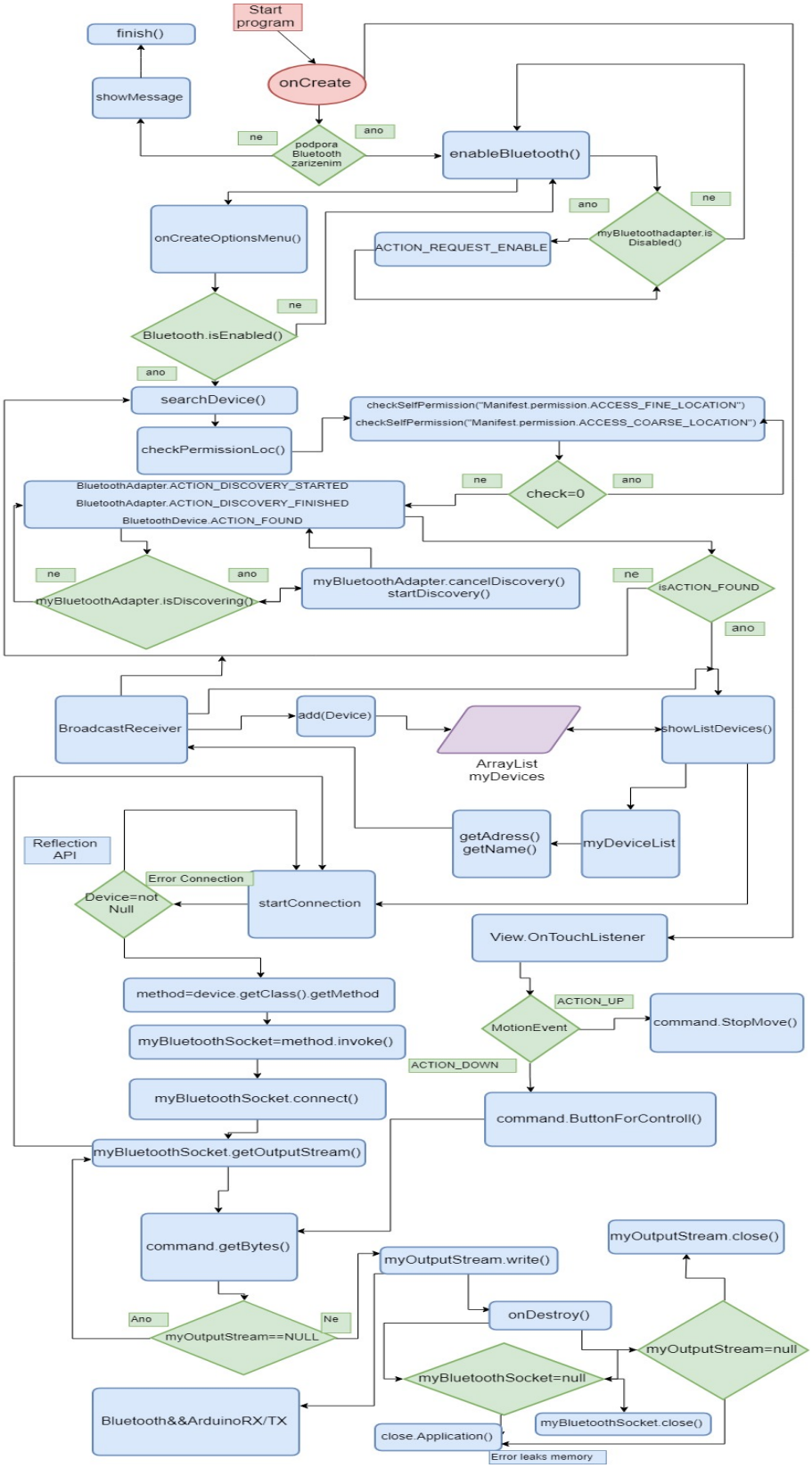
```
import android.content.Intent;
import android.os.Bundle;

import android.view.View;
import android.widget.ImageButton;

public class ChooseTypeConnectActivity extends Activity {
    private ImageButton imageButton;


    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.menu_choose_type_of_connect);
        imageButton = findViewById(R.id.imageButton2);
    }

    public void Back(View v) {
        Intent intent = new Intent(this, MainActivity.class);
        startActivity(intent);
    }

    public void nextActivity4(View v) {
        Intent intent = new Intent(this, ControlActivity.class);
        startActivity(intent);
    }
}
```

Tyto jednoduché třídy mají v programu dvě funkce:

- Oznámení uživatele, v jaké části programu se nyní nahází, například řízení probíhající přes Bluetooth nebo Wi-Fi apod.
- Splňují strukturu stromu a přepínaní mezi různými aktivity. Například aby se uživatel mohl lehce pohybovat pomocí grafického rozhraní mezi typy spojení a řízením různých akčních členů robota.

2.3.4 Třídy pro spojování a řízení

Nyní přicházím nejdůležitější část práce, a to vytvoření hlavní třídy pro řízení a vytvoření třídy pro seznam, ve kterém zapisuji jako objekty. Na obr.22 ukázaná celá struktura třídy, která slouží k lepšímu pochopení zdrojového kódu, který je dodán v příloze.

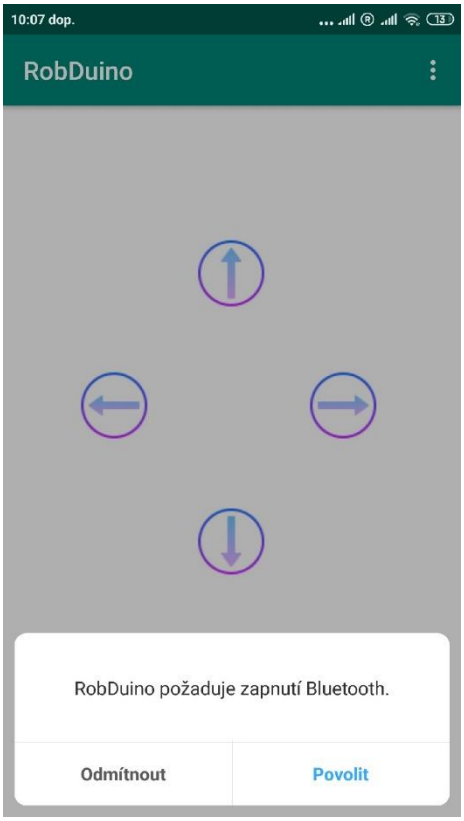


Obr. 22: Diagram struktury tříd pro spojovaní a řízení

Nyní mužů přijít na návrh struktury. Na začátku vytvořím třídu, ve které označím všechny proměnné, seznamy apod. Je dobré označovat také systémové Logy, které velmi pomohou při testu programu a vývoje. Vytvořím proměnnou typu TAG, která bude dostávat string hodnoty z logu, vytvořím proměnné typu bool pro všechny tlačítka a označím je jako false. Vytvořím proměnnou pro bluetooth adapter a poslední proměnná nyní bude REQUEST_ENABLE_BLUETOOTH, která slouží jako žádost. Tu použiji v metodě pro zapnutí bluetooth modulu. Nyní je potřeba vzít rodičovu metodu onCreate() a přepsat ji, zaprvé přiradím layout menu_Control.xml k třídě a také všechny obrázky pro řízení k proměnným a udělám kontrolu jestli obsahuje mobil bluetooth modul. Pokud zařízeni neobsahuje bluetooth, program automaticky přejde v menu do výběru spojení a bude volat metodu enableBluetooth().

```
if (myBluetoothAdapter == null) {
            Log.d(TAG, "onCreate: Your device does not support
bluetooth module");
            finish();
        }
        myDeviceList = new DeviceList(this, R.layout.device_item,
myDevices);

        //Turn on Bluetooth
        enableBluetooth();
    }
```

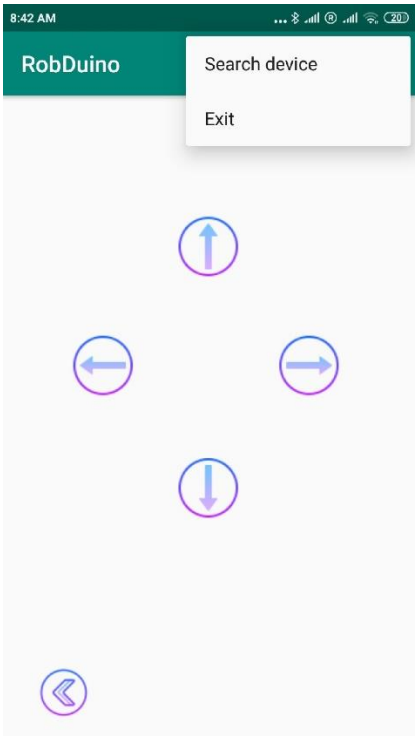
Přejdu na metodu enableBluetooth(), ve kterém udělám kontrolu zda je bluetooth zapnut či ne. Pokud není zapnut vytvořím objekt třídy Intent, který bude žádat u uživatele o zapnutí bluetooth modulu. Ukázka kódu je dále a na obr. 23 kde je vidět, jak žádá uživatele o zapnuti Bluetooth.



Obr. 23: Activity s požadavkem o zapnuti Bluetooth

```
private void enableBluetooth() {
        Log.d(TAG, "enableBluetooth()");
if (!myBluetoothAdapter.isEnabled()) {
Log.d(TAG, "enableBluetooth : Trying turn on Bluetooth");
Intent intent = new Intent(BluetoothAdapter.ACTION_REQUEST_ENABLE);
            startActivityForResult(intent, REQUEST_ENABLE_BLUETOOTH);
        }
    }
```

Dále budu přepisovat metodu onCreateOptionsMenu(), která slouží k zobrazování kontextového menu v Activity. Ukázka na obr. 24.



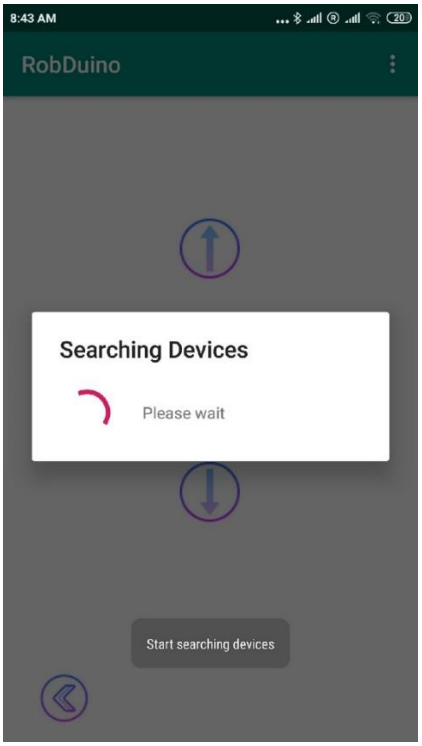
Obr. 24: Activity se zobrazením kontextového menu

Když uživatel zmáčkne tlačítko Search Device, tak v programu bude vyvoláná metoda searchDevice(), která volá předchozí metodu enableBluetooth(), aby zabránila chybě, při které dojde k hledání bluetooth bez zapnutého modulu. Druhá volaná metoda je checkPermissionLoc(), která kontroluje všechna povolení, která jsou ukázaná na obr. 19. Dále vytvořím nový objekt filtr, který bude filtrovat bezdrátové sítě, aby pracovaly jenom s Bluetooth a nikoliv s jinými bezdrátovými zařízeni. Pomocí příkazu „if" spustím proces vzdáleného vyhledávání zařízení. Níže je k nahlédnutí metoda searchDevice().

```
private void searchDevice() {
        Log.d(TAG, "searchDevice()");
        enableBluetooth();
        //must control Android version and give geo location(must to
be ,the rule from google)
        checkPermissionLoc();
        IntentFilter filter
= new IntentFilter(BluetoothAdapter.ACTION_DISCOVERY_STARTED);
        filter.addAction(BluetoothAdapter.ACTION_DISCOVERY_FINISHED);
        filter.addAction(BluetoothDevice.ACTION_FOUND);
        registerReceiver(myReceiver, filter);
```

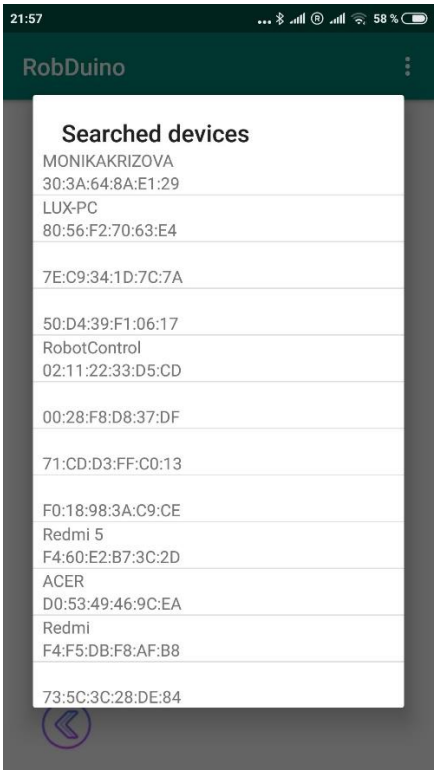
```
        if (!myBluetoothAdapter.isDiscovering()) {
            Log.d(TAG, "searchDevice :Start searching devices");
            myBluetoothAdapter.startDiscovery();

        }
        if (myBluetoothAdapter.isDiscovering()) {
            Log.d(TAG, "searchDevice :searching already
running...restarting");
            myBluetoothAdapter.cancelDiscovery();
            myBluetoothAdapter.startDiscovery();

        }

    }
```

Dále přejdu na broadcastReceiver, který zcela vyplňuje dvě funkce informování uživatelů o stavu programu. To znamená, že zobrazuje krátký text na dobu hledaní, a že probíhá hledání, nebo v případě vypadnutí bluetooth uživatel bude také informován. Druhá funkce probíhá, když najde bluetooth zařízení, které musí být zapsáno do seznamu, a to znovu dokud neskončí doba hledání. Ukázka je na obr. 25.



Obr. 25: Activity s oznámením uživatelé že prochází hledaní

Všechny zapsané Bluetooth moduly, budou ukázány pomocí metody showDeviceList(), která odkazuje na nově vytvořenou třídu DeviceList, takže metoda showDeviceList() odpovídá za uspořádání grafických prvků a práce s dalším xml souborem. Ukazuje povrch už běžící aktivity, ještě jednou se všemi vyhledanými v okolí zařízeni, ale data pro obsah tohoto objektu naplní nová třída. Na obr. 26 je ukázaná funkčnost metody showDeviceList() a třídy DeviceList. Je zde k nahlédnutí také část kódu nové třídy pro stažené adresy a jména okolních Bluetooth zařízení z objektů zapsaných do seznamu.



Obr. 26: Activity zobrazující seznam Bluetooth zařízení v okolí

```
    @SuppressLint("ViewHolder")
    @NonNull
    @Override
    public View getView(int position, View
convertView, @NonNull ViewGroup parent) {
        convertView = myLayoutInflater.inflate(myResourseView,null);

        BluetoothDevice device=myDevices.get(position);
        TextView tvName=convertView.findViewById(R.id.tvNameDevices);
        TextView
tvAddress=convertView.findViewById(R.id.tvAddressDevice);

        tvName.setText(device.getName());
```

```
        tvAddress.setText(device.getAddress());

        return convertView;
    }
}
```

Když začnu s návrhem datového proudu, musím vysvětlit, jak funguje reflexe v Java. Jedná se o mechanismus pro zkoumání dat programu, během jeho provádění. Reflexe umožní prozkoumat informace o polích, metodách a konstruktorech tříd. Samotný mechanismus reflexe umožňuje zpracovávat typy, které chybí během kompilace, ale které se objevily během provádění programu. Odraz a přítomnost celého logického modelu pro hlášení chybových informací umožňuje vytvořit správný dynamický kód [13].

Zde je základní seznam toho, co reflexe umožňuje:

- Naučit se / určit třídu objektu
- Získat informace o modifikátorech tříd, polích, metodách, konstantách, konstruktorech a nadtřídách
- Zjistit, které metody patří k implementovanému rozhraní / rozhraním
- Vytvořit instanci třídy s názvem třídy neznámé až do spuštění programu
- Získání a nastavení hodnoty pole objektu podle názvu
- Volání metody objektu podle názvu

Důvod proč bylo použita reflexe API, kvůli tomu, že metody z dokumentace google pomoci, které byly vytvořeny, nefungovaly. Kód pro vytvoření klientského připojení nechtěl běžet a vždy se vrátila chyba „Service discovery failed“.

```
BluetoothSocket bs=device.createRfcommSocketToServiceRecord(MY_UUID);
bs.connect();
```

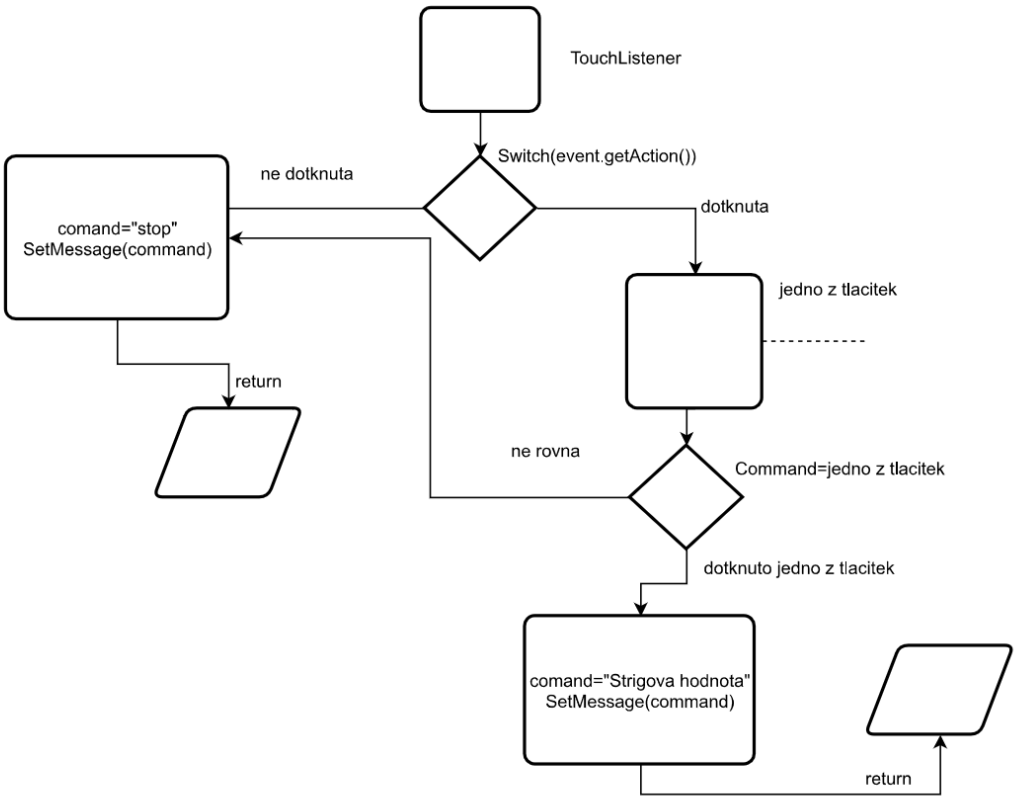
Tipem pro řešení tohoto problému bylo navrhování různých hodnot pro MY_UUID, to ale také nepomohlo, protože se mi nepodařilo najít správný UUID. A kvůli tomu jsem musel jít jinou cestou reflexe Api. Níže je ukázkový kód reflexe.

```
private void startConnection(BluetoothDevice device) {
   if (device != null) {
        try {
Method method=device.getClass().getMethod("createRfcommSocket", new
Class[]{int.class});
myBluetoothSocket = (BluetoothSocket) method.invoke(device, 1);

myBluetoothSocket.connect();

myOutputStream = myBluetoothSocket.getOutputStream();

                showMessage("Connection Successful");
            } catch (Exception e) {
                e.printStackTrace();
                showMessage("Error Connection");
            }

        }
    }
```

Funguje to tak, že získám požadovaný způsob podle jména a umožním k němu přístup. Volat objekt method, který využije invoke (Object, args), kde je Object – je stále jako instance třídy device. Args – Argumenty, tímto způsobem (skrze prvky pole) označuju typy parametrů metody, které můžu vyvolat prostřednictvím reflexe pro to, abych vytvořil nový socket během programu. Pro proud dat použiju funkci OutputStream, ke které přiřadím vytvořený socket.

Poslední část programu je sestavena z řídicích funkcí pro realizace, které vybral interface TouchListener, a který funguje na dotyk, což znamená, že když se dotknu (ACTION_DOWN podle diagramu) do nějaké části obrazovky, tak začne běžet proud dat a pokud displej bude uvolněn (ACTION_UP podle diagramu) bude běžet jiný proud dat, který bude posílat data do Arduino. V tuto chvíli Arduino musí pochopit poslaná data která přišla, a podle těchto dat začít správně reagovat. Jinak bude celá konstrukce rozdělena na Switch case, ve kterém mám funkce ACTION_UP a ACTION_DOWN. Uvnitř jednotlivých case jsem přiradil „if", do kterých jsem vložil všechny možné situace v případě řízení. Například aby nedošlo k situaci, kdy motory dostanou signály na otáčení dopředu a dozadu v jednou chvíli. Dole k nahlédnutí část konstrukce kódu pro řízeni a diagram (obr.27).

```
            switch (event.getAction()){
                //Buttons pressed condition
                case MotionEvent.ACTION_DOWN:
                    if(v.equals(leftImageButton)){

                        isLeftImageButton= !isLeftImageButton;
                        command = isLeftImageButton ? "l" : "s";
                        Log.d(TAG, "OnClick: isLeftImageButton = " +
isLeftImageButton);
                    }
```



Obr. 27: Diagram popisující funkčnost metod pro řízení

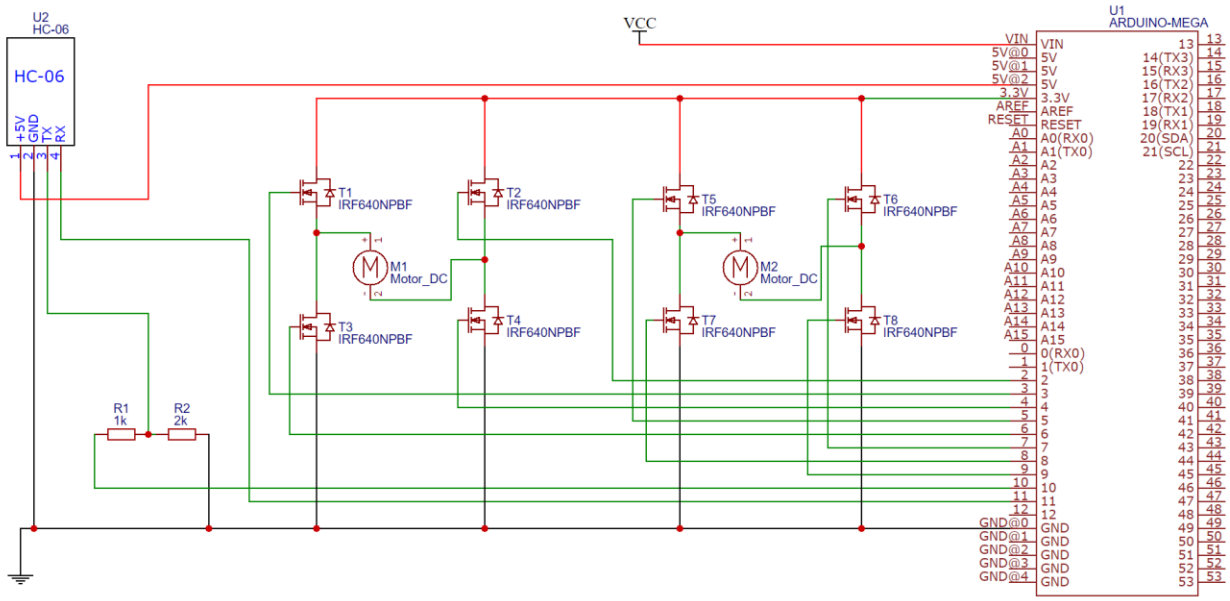
Aby nedošlo k úniku paměti na mobilu musím uzavřít socket nebo proud dat, a to všechno dám do metody OnDestroy, aby se proud dat ukončil jen při uzavírání nebo minimalizování activity.

3 Testování aplikací

Pro testování aplikací byl zprovozněn podvozek pro robota. Zaprvé navrhnu elektrický obvod, poté udělám montáž na podvozek laboratorního robota. Kvůli tomu, že Bluetooth zařízeni nejde testovat na virtuálním zdroji, budou testy provedeny na reálném zařízeni.

3.1 Návrh elektrického obvodu

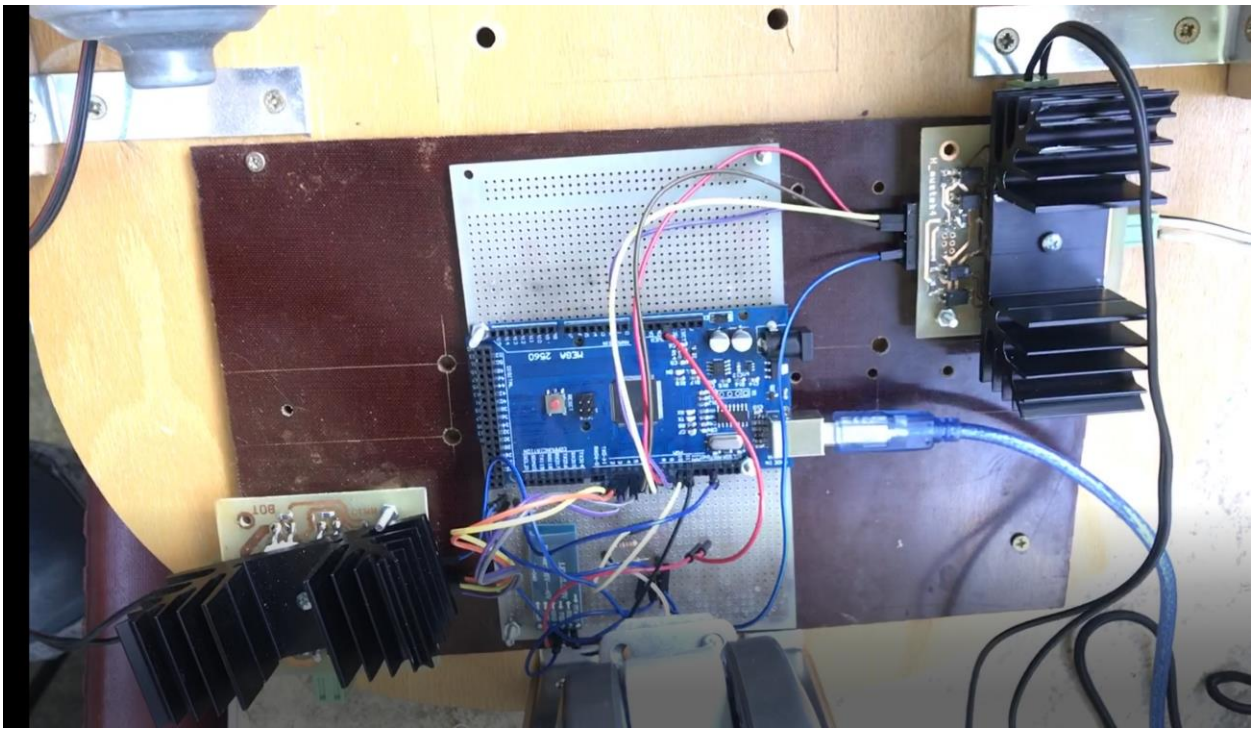
Celkové zapojení Arduino, H-můstku a BT HC 06 je znázorněné na obrázku 28. Funguje to tak, že BT HC 06 přijímá data a přeposílá je přes TX/RX komunikace, které budou následně zpracovány v procesoru Atmel Atmega 2560. Dále posílám z řídící desky jednotlivé signály na páry tranzistoru. Přes tyto páry tranzistorů T1, T4 a T5, T8 zajistím rovný pohyb. Přes jednotlivé páry T1, T4 zajistím pohyb doleva a pár T5, T8 pohyb doprava. Páry T2, T3 a T6, T7 odpovídají za reverse polarity v motorů, čímž umožňují zpětný chod motorů. Dělič napětí slouží pro TX pin, který pracuje v 3.3 V. A hlavní podmínkou pro obvod je, že Tranzistory musí vydržet proud v 3A a napětí v 12 V.



Obr. 28: Schéma zapojení

3.2 Montáž

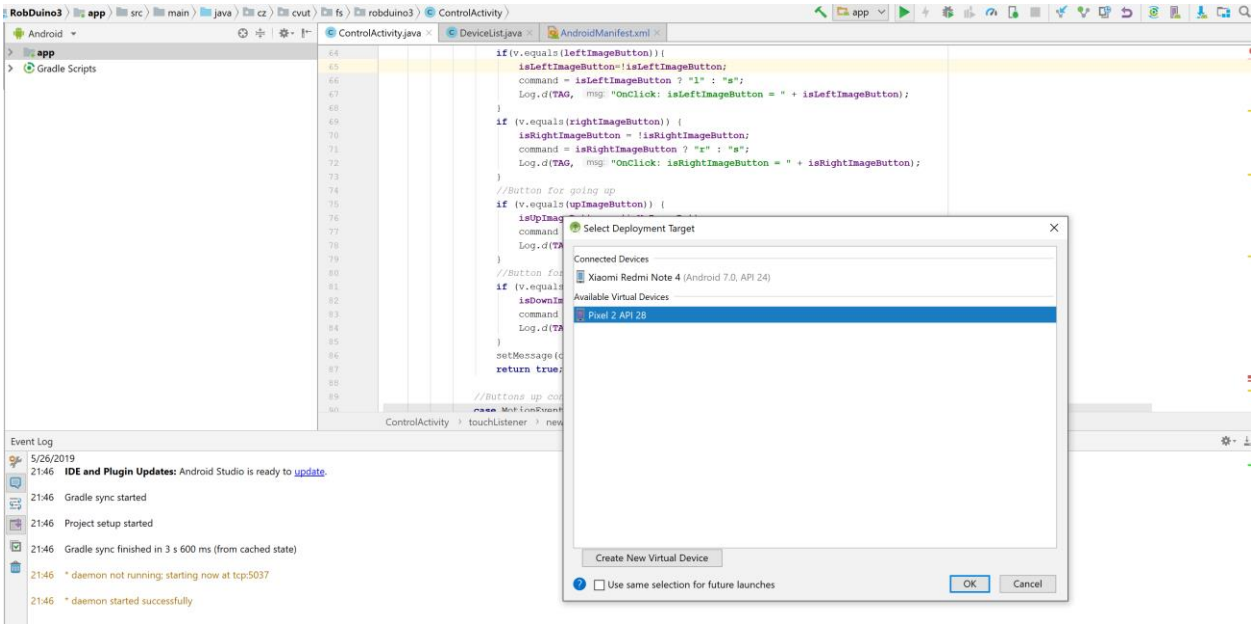
Postupně byl proveden montáž reálného zařízeni na laboratorního robota. Zaprvé jsem spojil na jedné společné desce Arduino s Bluetooth modulem a děličem napětí. Pak spojil Arduino a dva moduly, které spolu představují dvojitý H-můstek. Kvůli tomu, že přes H-můstek bude procházet velký proud musí odvádět teplo pomoci chladičů, které namontoval na jednotlivé H-můstky. Spojení chladiče s jednotlivým modulem uděláno pomocí šroubu a matice v centru desky modulu. Pak zkontroloval že po montáži chladičů v centru desky nedochází ke zkratu. Dále byli vyvrtány díry pro montáž desky na podvozek a celá deska byla namontovaná pomocí samořezných šroubů (obr.29).



Obr. 29: podvozek robota po montáži

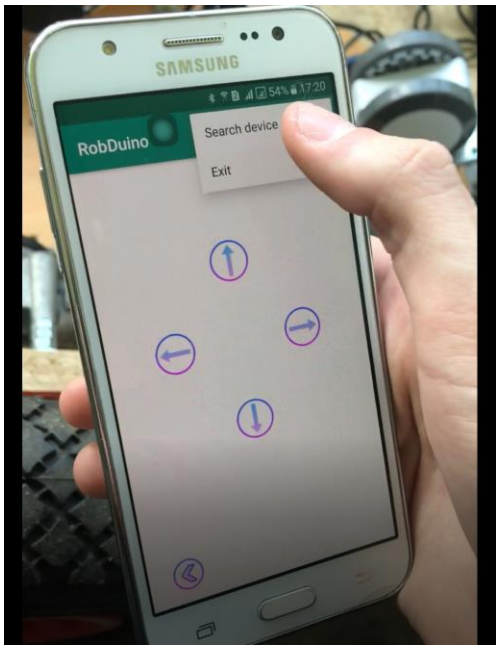
3.3 Testovaní aplikací

V Android studio lze testovat aplikace pomocí virtuálního zdroje, ve kterém můžu navrhnout parametry zařízení, které chci otestovat. Ale některé moduly, jako Bluetooth nelze otestovat pomocí virtuálního zařízení a tím pádem musím ladit program a dělat testy na reálném zařízení. Na obr. 30 znázorněno okno pro výběr reálného, a nebo virtuálního zařízení.



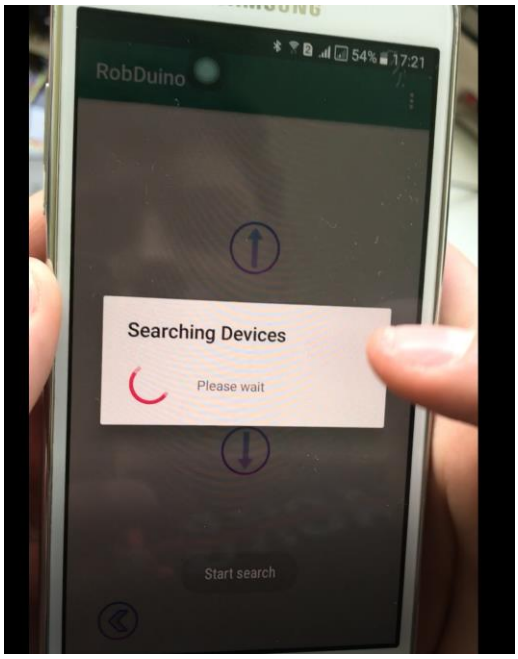
Obr. 30: Okno pro testovaní

Nahrání aplikací na reálné zařízeni prošlo úspěšně. Žádné chyby při kompilaci se nevyskytovali, ale to ještě neznamená, že se během testovaní logické chyby nevyskytnou. Pak postupně otestuji každou napsanou funkci, vidím že se všechny komponenty designu zobrazují správně.



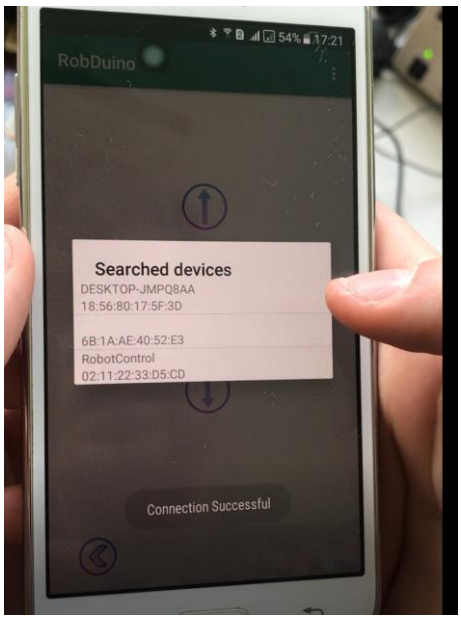
Obr. 31: Testovaní kontextového menu

Dál zkusím hledat zařízení kolem sebe, a vidím že animace pro hledání funguje dobře, a to je možné vidět na obr. 32.



Obr. 32: Testovaní funkčnosti animace

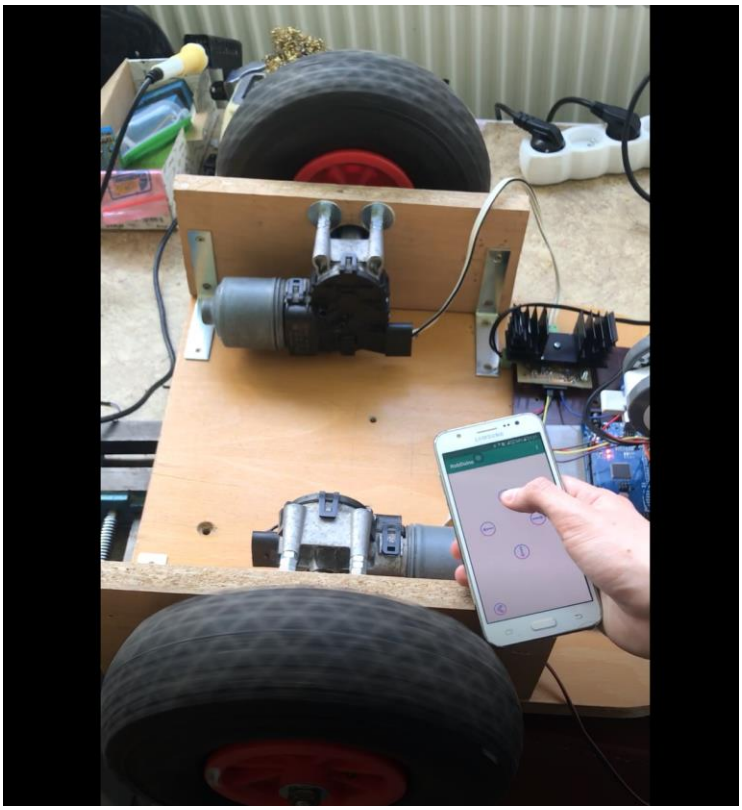
Nyní zkusím spojit s Arduino, a na obr. 33 je možné vidět že broadcastRecievier funguje a ukazuje, že spojení proběhlo úspěšně.



Obr. 33: Testovaní seznamu zařízení

Jak lze vidět na obr. 34, Arduino přijímá signál z mobilu a správně ho interpretuje. Ale vyskytlo se tam několik chyb. První byla v tom, že levé kolo na začátku nefungovalo,

a to bylo způsobeno logickou chybou, že levé tlačítko na mobilu mělo pořád false. To znamenalo, že ve funkci pro levé tlačítko chyběla negace, další chyba už byla způsobená Bluetooth modulem. Ve chvíli kdy se dotýkám jednoho z tlačítek a poté v ten samý čas tlačítko pustím, do Arduina dojde špatný balíček dat, a nebo pokud vypadne spojení může dojít ke stavu, kdy kolo prodlouží otáčení. Takže jsem rychle navrhnul opravu, a to takovou, že do Arduino jsem přidal ještě pár řádků kódů, který jsem před zpracováním balíčku dat zkontroluji jestli je Bluetooth spojen s dalším zařízením. A aby to bylo ještě více zabezpečeno, přidal jsem další řádky kódu, který posílají log 0 na tranzistory, když nedostává signál za dobu 3 s. Testoval jsem to pět krát po dobu 3 minut, a další chyby se nevyskytovaly.



Obr. 34: Testovaní funkčnosti řídích signálů

4 Návrh na zlepšení

Prvním důležitým bodem, při práci s Bluetooth je to, že programátor musí vědět, že datové balíčky mohou být poškozeny pří komunikaci [14]. Navíc mohou být zašuměny. Takové jevy se vyskytují poměrně často a musí být předzpracovány. Dalším problémem je, že Bluetooh připojení je málokdy statické.

Jako příklad řešení těchto problémů jsem navrhl jednoduchý kód pro zpracování událostí, které jsou definované prvním počtem N-bajtu, a ověřené pomocí jednoduchého kontrolního součtu [15]. Všechny balíčky jsou rozděleny do dvou typů: se statickou a dynamickou délkou, přenášenou samostatným bytem.

Základní struktura programu je znázorněná dole.

```
sealed class Event {
val headSize: Int = 2
abstract val head: ByteArray
abstract fun isCorrupted(): Boolean
}
```

Dále, když jsem definoval sady konstantních vlastností pro všechny balíčky, bylo nutné nějakým způsobem formalizovat podmínky, za kterých:

- Rozhodnu, zda balíček patří nějakému typu;
- Musí být přidán do buffer, jelikož balíček není kompletní;
- Musí být vyčištěn buffer, vůči tomu že nebyli splněné podmínky pro uložení dat do bufferu;
- Musí být sestaven balíček z bufferu s následnou verifikaci.

Tyto čtyři podmínky vedou k následujícímu rozhraní:

```
interface EventMatcher {

    val headSize: Int

    fun matches(packet: ByteBuffer): Boolean

    fun create(packet: ByteBuffer): Event

    fun shouldBuffer(packet: ByteBuffer): Boolean

    fun shouldDrop(packet: ByteBuffer): Boolean
}
```

Vytvořím komponentu Proxy rozhraní, ve které budou jednoduché funkce přiřazeny.

```
 class EventMatchersAdapter {

    private val matchers = mutableMapOf<KClass<out Event>,
EventMatcher>()

    fun register(event: KClass<out Event>, matcher: EventMatcher)
            = apply { matchers.put(event, matcher) }

    fun unregister(event: KClass<out Event>)
            = apply { matchers.remove(event) }

    fun knownEvents(): List<KClass<out Event>>
            = matchers.keys.toList()

    fun matches(packet: ByteBuffer, event: KClass<out Event>): Boolean
            = matchers[event]?.matches(packet) ?: false

    fun shouldBuffer(packet: ByteBuffer, event: KClass<out Event>):
Boolean
            = matchers[event]?.shouldBuffer(packet) ?: false

    fun shouldDrop(packet: ByteBuffer, event: KClass<out Event>):
Boolean
            = matchers[event]?.shouldDrop(packet) ?: false

    fun create(packet: ByteBuffer, event: KClass<out Event>): Event?
            = matchers[event]?.create(packet)
}
```

V balíčcích popisuji způsob určení, zda byl balíček poškozen nebo ne, a to pomocí Kontrolního součtu bytu. Matematicky je kontrolní součet výsledkem hashovací funkce použité k výpočtu kontrolního kódu – malého počtu bitů uvnitř velkého datového bloku, pro můj příklad balíček dat z Arduino by se měl dostat do mobilu, který se používá k detekci chyb během přenosu nebo ukládání informací [15]. Hodnota kontrolního součtu se přidá na konec datového bloku bezprostředně před začátkem přenosu nebo záznamu dat na libovolné paměťové médium. Následně je zkontrolováno, zda byla potvrzena integrita dat. K nahlédnutí je jednoduchý algoritmus odpovídající (1) pro to, když dostávám staticky balíček dat.

$$CRC = byte(1) + byte(2) + byte(3) + \ldots + byte(N)(1)$$

Pro dynamický balíček dat ještě musí být kód doplněn pro rozšíření rozměrů na dva byte. Na začátku ještě jeden byte odpovídající za délku, plus ještě jeden byte od kontrolního součtu.

Dále potřeba dopsat ještě algoritmus čtení dat, který bude:

- Pracovat s různými typy dat;
- zničit poškození balíčky;
- podporovat spojení.

Ukázka, jak by mohl vypadat algoritmus v kódu je v příloze pod názvem (EventBridge) a funguje tak, že každý balíček dat který dostavám s Arduino porovná s délkou balíčku na mobilu, a pokud nebudou rovny, tak vyčistí buffer. To samé provedu s kontrolním součtem. V tomto příkladu bych chtěl ukázat, jak snadné je udržet modularitu při zpracování téměř libovolných událostí. Mimochodem, nemusí nutně pocházet ze zdroje Bluetooth (dosud nebyl žádný kód závislý na technologii Bluetooth) a zároveň se vyhnout možnému poškození dat a filtrace šumu komunikačního kanálu. Jinak napsaný kód v příloze slouží jako příklad pro realizace filtru pro Bluetooth, který slouží pro podporu statického spojení.

Další možné zlepšení je přidat další bezdrátové typy komunikací, a to buď Wi-Fi anebo mobilní síť. Když spojení přes Wi-Fi bude podobné jako bluetooth ohledně návrhu a napsání kódu, a budou tam pouze použity jiné knihovny pro spojení, ale celkově to bude dost podobná úloha, při dodání možností funkce prací přes mobilní sítě bude potřeba zlepšit aplikace na klient-server ve kterém:

- Server je druh programu běžícího na vzdáleném počítači a implementující funkcionalitu „komunikace" s klientskými aplikacemi (naslouchá požadavkům, rozpoznává přenášené parametry a hodnoty, správně na ně reaguje) [16];
- klient – v našem případě program na mobilním zařízení, které může generovat na požadavek, který server chápe a posílá zpětnou odpověď [16];
- rozhraní interakce – určitý formát a způsob odesílání / přijímání žádostí / odpovědí oběma stranami [16].

Byly v roce 2013 vyvíjeny nové knihovny Volley a Retrofit, a při vytvoření aplikací fungující jako klient-serverová, bych chtěl doporučit jejich použití. Dále vysvětlím jejich výhody. Volley je obecnější knihovna určená pro vytváření sítí, zatímco Retrofit je speciálně navržen pro práci s REST Api [17]. Také to byla poslední knihovna, která se stala univerzálně uznávaným standardem ve vývoji aplikací typu klient-server.

Retrofit v porovnání s jinými způsoby, má některé hlavní výhody:

- Velmi pohodlné a jednoduché rozhraní, které poskytuje plnou funkčnost pro provádění jakýchkoli požadavků;

- Flexibilní konfigurace – pro splnění požadavku můžete použít libovolného klienta, libovolnou knihovnu pro analýzu json atd.;
- Není třeba samostatně provádět parsování Json – tuto práci provádí Jsonova knihovna (ale nejen Json);
- Pohodlné zpracování výsledků a chyb;
- Rx podpora, která je dnes také důležitým faktorem.

5 Závěr

Na začátku byla provedena rešerše na téma mobilní aplikace v průmyslu a hobby sektoru, ze které lze vidět, že v dnešní době se výrobci starají o dodávku mobilních technologií, pro lepší spravováni svých produktů. Vzhledem k tomu, že klasifikace těchto řešení dosud neexistuje, bylo vyvinuto vlastní rozdělení, založené na vztahu aplikace k průmyslovému procesu a jeho umístění v architektuře průmyslového řídicího systému.

Bylo uvedeno rozdělení mezi průmyslové aplikace a aplikace pro hobby sektor. Následně byl proveden návrh vlastní aplikace pro řízení laboratorního robota, ve které bylo odůvodněno, proč byla vybraná jako platforma realizace Android a proč jako programovací jazyk byla vybraná Java. Dalším krokem bylo prozkoumaní základů programovaní v Android a práce s Android Studio. Poté začal návrh celkové struktury programu a bylo vysvětleno proč a jaké knihovny byly použity.

Modularita byla vyřešena tak, že celková struktura aplikace připomíná strom a pro přidání například dalšího akčního prvku nebo jiného typu spojení, potřeba vytvořit další větev která může být zcela nezávislá na předchozím kódu. Například když potřebuju vytvořit další větev s novým motorem mám dvě možnosti, první okopírovat část kódu pro připojování k Bluetooth modulu a proudu dat z Bluetooth modulu do nové větve. Druhý způsob je o něco těžšího, ale umožňuje ušetřit paměť, a to za pomocí použití známé reflexe.

Dalším krokem bylo zprovoznění podvozku pro robota, na kterém byla otestovaná aplikace, celkový obvod prokázal schopnost správně přijímat a interpretovat balíčky dat z mobilní aplikace. Po testech bylo identifikováno několik chyb. Důležitá chyba byla pouze jedna a to ta, že pokud nastává stav vypadnutí signálu při dotyku na display, motory se můžou nadále otáčet. Vzniklé chyby byly následně upraveny. Dále aplikace neprokázala kritické chyby a fungovala tak, jak byla zamýšlena.

Prvním bodem pro následné zlepšení, by mělo být zlepšení funkčnosti Bluetooth. Za tímto účelem byl proveden předpoklad a pokus pro zlepšení. Byl navrhnut interface, který předpokládá budoucí zlepšení ve filtraci poškozených balíčků dat, zavedení zapracování dalších datových typů, podporu stálého spojení kvůli algoritmu kontrolního součtu. V dalších bodech by se měl program rozšířit na další typy bezdrátových sítí jako je Wi-Fi apod. Byl stručně popsán způsob možností rozšíření aplikace. Pro další použití je zamýšleno vyzkoušet program s průmyslovými kontroléry, které je možno propojit s novějšími Bluetooth moduly a vyzkoušet, jak budou komunikovat mezi sebou, i spolu s mobilní aplikací. Dalším postupem je výzkum možností spolehlivě a úspěšně řídit průmyslové kontroléry z mobilního zařízení.

Literatura

[1] *Produkty od schneider electric* [online], [cit. 2019-04-1]. Dostupné z: https://automatizace.hw.cz/industry-40-s-produkty-od-schneider-electric.html

[2] *PRŮMYSLOVÉ ŘÍDICÍ SYSTÉMY* [online], [cit. 2019-04-2]. Dostupné z: Http://projekty.fs.vsb.cz/463/edubase/VY_01_022/Pr%C5%AFmyslov%C3%A9%20a%20%C5%99%C3%ADdic%C3%AD%20syst%C3%A9my.pdf [online], [cit. 2019-05-28]. Technická univerzita Ostrava...

[3] *Řízení malých robotů KUKA díky aplikaci robotu prostřednictvím smartphonu* [online], [cit. 2019-04-6]. Dostupné z: https://www.kuka.com/cs-cz/odv%C4%9Btv%C3%AD/solutions-database/2017/11/solution-robotics-robomotion

[4] *Pro-face Remote HMI* [online], [cit. 2019-04-8]. Dostupné z: https://play.google.com/store/apps/details?id=com.proface.remotehmi&hl=en

[5] *OPC XML DA* [online], [cit. 2019-04-8]. Dostupné z: https://www.apkmonk.com/app/com.kassl.dOPCMExplorer/

[6] *Aplikace od Ar.Drone* [online], [cit. 2019-04-12]. Dostupné z: https://www.svetandroida.cz/kvadrokoptera-ar-drone-ovladnete-vzduh-skrz-sveho-androida/

[7] *DJI Discovery* [online], [cit. 2019-04-15]. Dostupné z: https://www.airmap.com/airmap-for-drones-app-fly-dji/

[8] *Developer Apple* [online], [cit. 2019-04-20]. Dostupné z: https://developer.apple.com/library/archive/documentation/NetworkingInternetWeb/Conceptual/CoreBluetooth_concepts/AboutCoreBluetooth/Introduction.html

[9] *Developer Anroid* [online], [cit. 2019-04-20]. Dostupné z: https://developer.android.com

[10] *Most popular mobile operation system* [online], [cit. 2019-04-24]. Dostupné z: https://www.webopedia.com/DidYouKnow/Hardware_Software/mobile-operating-systems-mobile-os-explained.html

[11] *Most Popular Technologies* [online], [cit. 2019-05-1]. Dostupné z: https://insights.stackoverflow.com/survey/2018/#technology

[12] *Programming language for Android* [online], [cit. 2019-05-2]. Dostupné z: https://www.androidauthority.com/develop-android-apps-languages-learn-391008/

[13] *Oracle tutorial for Java developing* [online], [cit. 2019-05-5]. Dostupné z: https://docs.oracle.com/javase/tutorial/reflect/

[14] *BEZDRÁTOVÉ A MOBILNÍ KOMUNIKACE* [online], [cit. 2019-05-10]. Dostupné z: http://data.idnes.cz/soubory/tec_checktech/a041123_jma_skr_bezdratove-mobilni-komunikace.pdf

[15] *Checksum algorithms* [online], [cit. 2019-05-15]. Dostupné z: https://computer.howstuffworks.com/encryption7.htm

[16] *Client server communication* [online], [cit. 2019-05-28]. Dostupné z: https://www.ibm.com/support/knowledgecenter/en/SSGU8G_12.1.0/com.ibm.admin.doc/ids_admin_0123.htm

[17] *Android REST APIs* [online], [cit. 2019-05-28]. Dostupné z: https://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=8618824

Použité nástroje

Android Studio IDE Dostupné z: https://developer.android.com/studio

Vytvoření digramu Dostupné z: https://www.draw.io/

Arduino IDE Dostupné z: https://www.arduino.cc/en/Main/Software

Seznam příloh

Příloha 1: RobDuino.apk (zkompilována aplikace do formátu apk)

Příloha 2: RobDuino (ve formátu zdrojového kódu)

Příloha 3: Návrhový třídy a interface (Ukázkový soubor tříd pro zlepšeni aplikace)