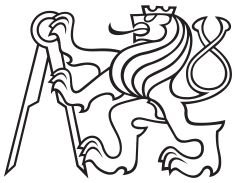
Diplomová práce



České
vysoké
učení technické
v Praze

F3

Fakulta elektrotechnická
Katedra radioelektroniky

RC model auta s částečnou autonomií

Bc. Filip Křemen

Vedoucí: doc. Ing. Stanislav Vítek, Ph.D.
Zaměření: Technologie internetu věcí
Květen 2024

ZADÁNÍ DIPLOMOVÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Křemen** Jméno: **Filip** Osobní číslo: **474717**

Fakulta/ústav: **Fakulta elektrotechnická**

Zadávající katedra/ústav: **Katedra radioelektroniky**

Studijní program: **Elektronika a komunikace**

Specializace: **Technologie internetu věcí**

II. ÚDAJE K DIPLOMOVÉ PRÁCI

Název diplomové práce:

RC model auta s částečnou autonomií

Název diplomové práce anglicky:

RC Model Car with Partial Autonomy

Pokyny pro vypracování:

1) Navrhněte a implementujte dálkově řízený model auta, které bude schopno částečné autonomie pro případ, že dojde ke ztrátě spojení s řídicí aplikací.
2) Pro komunikaci s vozítkem využijte technologii BLE. Řízení bude probíhat prostřednictvím mobilní aplikace.
3) Vozítko má vlastní výpočetní jednotku pro zpracování dat ze senzorů a řízení motorů. Dále je vybaveno kamerou a lidarem.
4) Vozítko otestujte v reálném provozu a diskutujte případné nedostatky.

Seznam doporučené literatury:

[1] WHITE, Elecia. Making Embedded Systems: Design Patterns for Great Software. " O'Reilly Media, Inc.", 2011.
[2] NOVIELLO, Carmine. Mastering STM32 book. 2019.
[3] LEA, Perry. Internet of Things for Architects: Architecting IoT solutions by implementing sensors, communication infrastructure, edge computing, analytics, and security. Packt Publishing Ltd, 2018.

Jméno a pracoviště vedoucí(ho) diplomové práce:

doc. Ing. Stanislav Vítek, Ph.D. katedra radioelektroniky FEL

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

Datum zadání diplomové práce: **09.02.2024** Termín odevzdání diplomové práce: **24.05.2024**

Platnost zadání diplomové práce: **21.09.2025**

doc. Ing. Stanislav Vítek, Ph.D. podpis vedoucí(ho) práce	doc. Ing. Stanislav Vítek, Ph.D. podpis vedoucí(ho) ústavu/katedry	prof. Mgr. Petr Páta, Ph.D. podpis děkana(ky)

III. PŘEVZETÍ ZADÁNÍ

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

. Datum převzetí zadání

Podpis studenta

Poděkování

Chtěl bych poděkovat panu doc. Ing. Stanislavu Vítkovi, Ph.D. za trpělivost a komunikaci při vypracování této práce. Děkuji také rodině a přátelům za podporu při psaní této práce.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací.

V Praze dne 19. května 2024,

Abstrakt

Práce se zabývá návrhem měřicí a řídicí jednotky pro RC model, který je ovládaný mobilní aplikací pomocí protokolu Bluetooth Low Energy. V úvodní části je popsán návrh systému a protokol Bluetooth Low Energy. Dále pak práce popisuje schéma a desku plošných spojů jednotky. V další části je popsán firmware pro ovládací jednotku a aplikace pro mobilní telefon ve frameworku Flutter. V poslední části se práce zabývá detekcí čar v obraze a řízením RC modelu uvnitř těchto čar. V závěru je zhodnocení navrženého systému.

Klíčová slova: Bluetooth Low Energy, STM32WB, Flutter, OpenCV, RF

Vedoucí: doc. Ing. Stanislav Vítek, Ph.D.

Abstract

The thesis deals with the design of a measurement and control unit for an RC model which is controlled by a mobile application using Bluetooth Low Energy protocol. In the introduction, the design of the system and the Bluetooth Low Energy protocol are described. The thesis then describes the schematic and circuit board of the unit. Then the thesis describes the firmware for the control unit and the mobile phone application written in the Flutter framework. The last part of the paper deals with the detection of lines in the image and the control of the RC model inside these lines. Finally, an evaluation of the implementation of the overall system is presented.

Keywords: Bluetooth Low Energy, STM32WB, Flutter, OpenCV, RF

Obsah

Obrázky

Tabulky

Kapitola 1

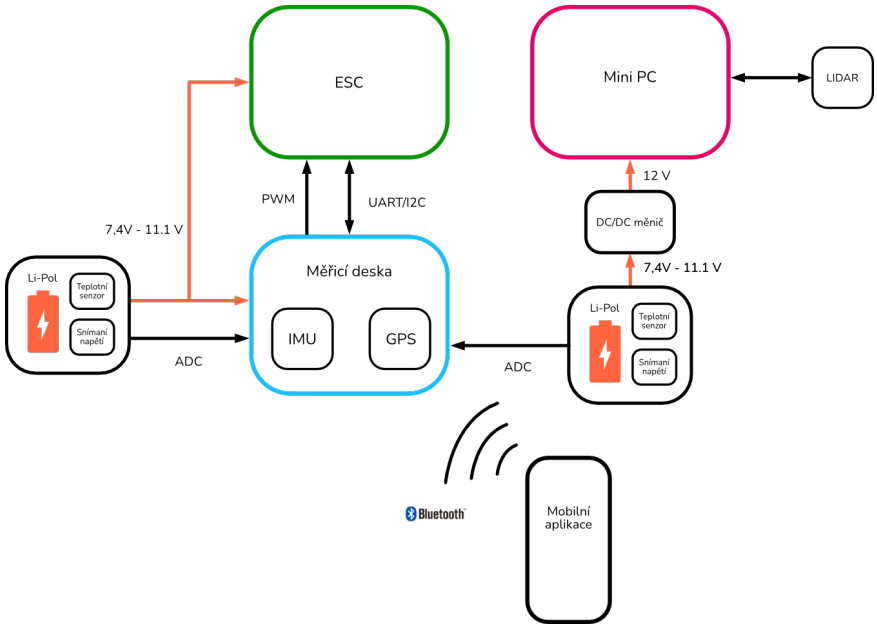
Úvod

Předmětem této práce je navrhnout řídicí a měřicí systém pro RC model buginy, který bude dálkově řízen pomocí mobilní aplikace a protokolu Bluetooth Low Energy. Model je v měřítku 1:12 a je poháněn 2847-3200KV třífázovým bezkomutátorovým motorem. Dále model obsahuje 3.5 kg servomotor. Model je původně řízen 2.4GHz čtyřkanálovým vysílačem s dosahem 150 m, který nastavuje šířku pulzů PWM modulace pro ESC regulátor a servomotor.

Obrázek 1.1: RC model buginy - Rlaarlo AM-X-12 [1]

1.1 Návrh systému

Celý systém je napájen pomocí dvou Li-Pol akumulátorů. Jeden sloužící pro ESC (Electronic Speed Controller) a měřicí desku, druhý je pak určený pro mini PC. Každý z akumulátorů bude opatřen teplotním senzorem a dále se bude na každém akumulátoru snímat elektrické napětí, aby nedošlo ke zničení článků kvůli podpětí a celý systém se včas mohl vypnout. Teploty a elektrického napětí akumulátorů se bude snímat na pinu mikrokontroléru pomocí ADC převodníků v měřicí desce. Napětí z akumulátorů bude dále sníženo na potřebné napájecí napětí mikrokontroléru pomocí DC/DC měničů, případně LDO regulátorů. PWM signál z měřicí desky je připojen na vstup ESC, který bude řídit rychlost otáčení bezkomutátorového motoru. Získaná data ze senzorů by z měřicí desky pomocí Bluetooth Low Energy byla zobrazována v mobilní aplikaci, která bude napsána pomocí frameworku Flutter. Dále by systém obsahoval mini PC komunikující s lidarem a webkamerou. Počítač obsahuje procesor Intel Celeron N100 s 16 GB LPDDR5 RAM pamětí a SSD o kapacitě 512 GB. Celkové rozměry jsou 89.4 mm x 89.4 mm x 43.5 mm, které jsou podobné jako např. u Raspberry Pi, přitom ale nabízí daleko vyšší výkon při podobné ceně. Hmotnost činí 207.3 g. Na počítači je nainstalovaný Linux s distribucí Ubuntu, na kterém bude možné zpracovat obraz z webkamery nebo lidaru. Mini PC bylo zvoleno z důvodu nedostatku Raspberry Pi 5 na trhu a bylo zvoleno jako vhodná alternativa s výhodou toho, že se jedná o běžný x86 systém.



Obrázek 1.2: Návrh systému

1.2 Bluetooth Low Energy

První verze komunikačního protokolu Bluetooth byla vydána v roce 1994 a nabyla velké popularitě. Najdeme ho v současné době v mnoho elektronických zařízení jako jsou mobilní telefony, chytré hodinky, počítače, bezdrátové sluchátka, reproduktory a mnoho dalšího. Protokol má dvě verze Bluetooth Classic a Bluetooth Low Energy. První z nich se používá v bezdrátových reproduktorech, informačních systémech pro automobily a headsetech. Bluetooth Low Energy byla představena ve verzi 4.0 a najde uplatnění všude, kde je požadována nízká spotřeba energie a nebo při sběru dat o malých objemech. Ačkoliv obě verze nesou stejný název, jsou mezi sebou nekompatibilní. Existují zařízení, kde jsou implementovány obě verze např. chytré telefony nebo počítače, které pracují v tzv. dual módu. [2]

1.2.1 Rozdíly mezi verzemi Bluetooth

Mezi verzemi jsou značné rozdíly, a proto nejsou schopné mezi sebou napřímo komunikovat. Tabulka níže popisuje některé z nich.

Bluetooth Classic	BLE
Nevhodné pro low-power aplikace	Vhodné pro low-power aplikace
Streamování audia a přenos souborů	Vhodné pro malé objemy dat
79 kanálů	40 kanálů

Tabulka 1.1: Rozdíly mezi Bluetooth Classic a BLE [2]

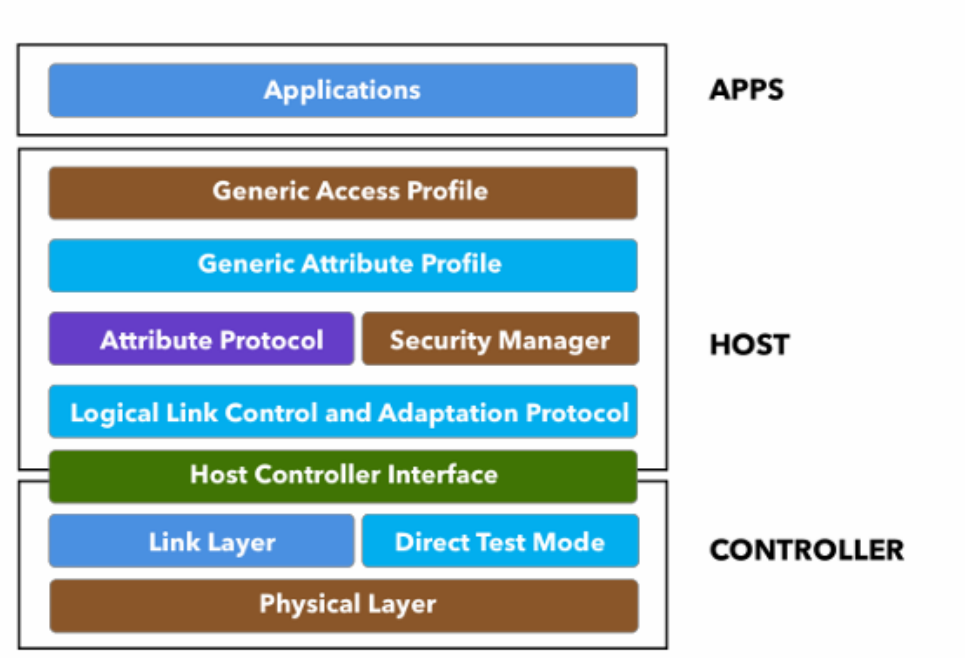
Datový tok BLE je omezený u verze 4.2 na 1 Mbps. U verze 5 a vyšší může být až 2 Mbps. Protokol disponuje i long-range funkcionalitou, kde ale datový tok klesá k 500 Kbps nebo až ke 125 Kbps. Jedná se datový tok na fyzické vrstvě. Výsledná rychlost, kterou zaznamená koncový uživatel, je nižší z těchto důvodů. [2]

- Časové mezery mezi pakety - Ve specifikaci je definové časové okno o hodnotě 150 μs, ve kterém nejsou zasílána žádná data.
- Pakety obsahují další informace např. pro řízení komunikace a kontrolu dat.
- Některé pakety mohou přijít porušené, a tak je nutné je poslat znovu.

Mezi hlavními výhodami je nízká spotřeba, která umožňuje použití protokolu v zařízeních, která jsou napájena bateriemi. Další výhodou je, že specifikace protokolu jsou zdarma volně dostupné, což snižuje náklady při vývoji. Dále nízká cena modulů a rozšířenost v mobilních telefonech je asi hlavní výhodou oproti ostatním protokolům jako ZigBee, Z-Wave nebo Thread.[2]

1.2.2 Architektura

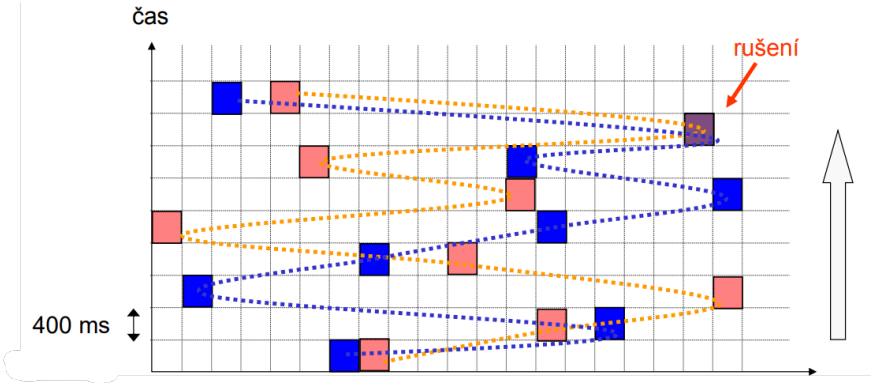
Architektura protokolu se dělí na hlavní 3 části - application, host a controller.



Obrázek 1.3: Architektura protokolu BLE [2]

1.2.3 Fyzická vrstva

Protokol pracuje ve frekvenčním pásmu 2.4 GHz a je rozdělen na 40 rádiových kanálů, jejichž středy jsou od sebe vzdáleny 2 MHz. Tři z těchto kanálů slouží jako primární kanály pro advertising. Ostatní pak slouží jako sekundární kanály pro advertising a pro přenos dat. Pro komunikace se využívá tzv. frequency hopping spread spectrum (FHSS), kdy komunikující zařízení přepíná náhodně svůj rádiový kanál. To vede k větší spolehlivosti a pomáhá snižovat vzájemného rušení při více zařízení v okolí. Pokud se zařízení vzájemně ruší, tak jen krátkou danou dobu, ze které lze komunikaci obnovit. [3]

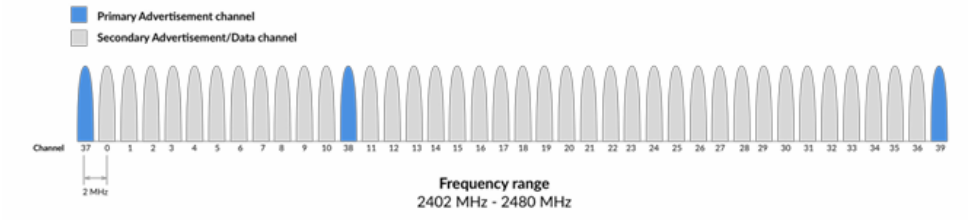


Obrázek 1.4: Frequency hopping [3]

Vysílací výkony jsou následující.

- Maximální vysílací výkon - 100 mW (+20 dBm) pro verzi 5 a vyšší, 10 mW (+10 dBm) pro verzi 4.2 a nižší
- Minimální vysílací výkon - 0.01 mW (-20 dBm)

Starší verze 4.0 až 4.2 mají datový tok fixní na 1 Mbps. Změna přišla ve verzi 5, kde je možnost si vybrat mezi 2 Mbps PHY, tedy dvojnásobné rychlosti a nebo Code PHY, která určena pro vzdálenou komunikaci (okolo 1 km). [2]

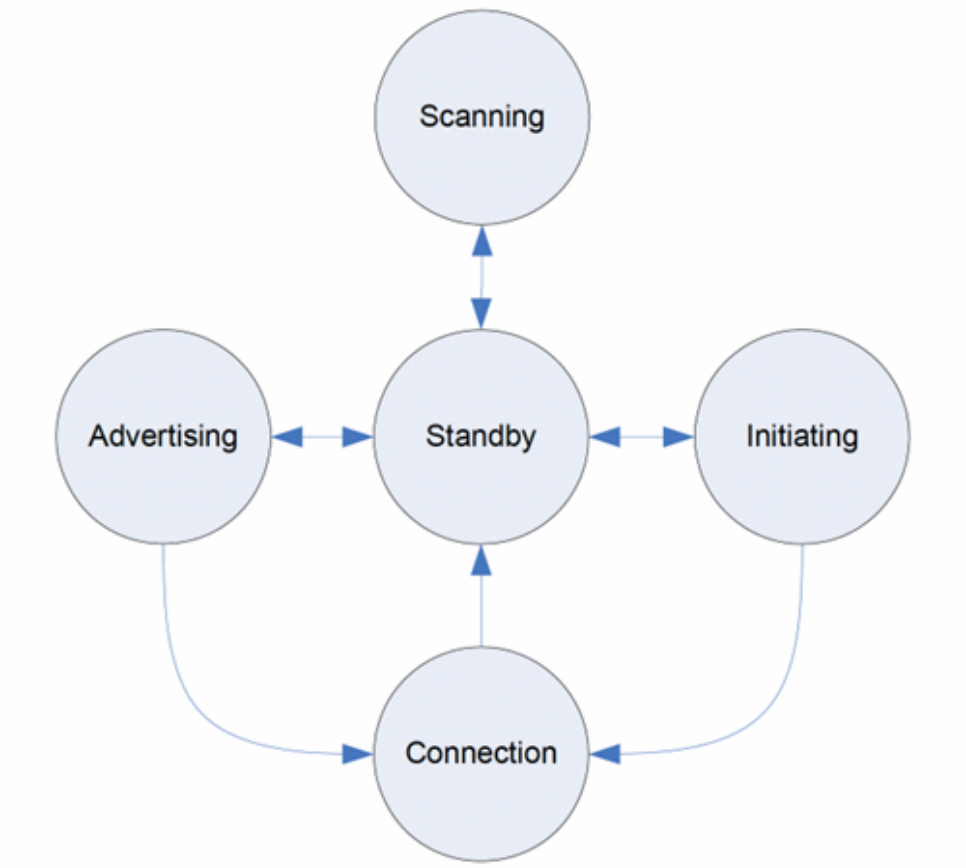


Obrázek 1.5: Rádiové kanály BLE [2]

1.2.4 Linková vrstva

Tato vrstva slouží pro komunikaci mezi fyzickou vrstvou a vyššími vrstvami. Stará se o požadované časování pro splnění BLE specifikace. Dále se stará o operace jako CRC, generování náhodných čísel a šifrování. Tři hlavní stavy, ve kterém zařízení může pracovat, jsou advertising, scanning a connected. Pokud je zařízení ve stavu advertising, umožňuje ostatním zařízení, které skenují okolí, aby se k sobě připojila. Pokud zařízení, které skenuje, zažádá o připojení, tak se obě zařízení dostanou do stavu connected, tedy do stavu kdy jsou připojeny. [2]

- Standby - Základní stav, kdy rádio nepřijímá ani nevysílá žádná data.
- Advertising - Zařízení vysílá pakety pro advertising pro ostatní zařízení.
- Scanning - Zařízení hledá zařízení, která jsou ve stavu advertising.
- Initiating - Zařízení se rozhodlo připojit se k dalšímu zařízení.
- Connected - Zařízení jsou připojena, probíhá přenos dat.



Obrázek 1.6: Stavy linkové vrstvy [2]

Adresování

Protokol Bluetooth má 48-bitové adresy, které se dělí na veřejné a náhodné adresy. Veřejné adresy jsou fixní a jsou dány při výrobě zařízení. Musí být registrovány u IEEE, podobně jako u WiFi nebo Ethernet MAC adres. Výrobce si může zvolit, jestli použije veřejnou a nebo náhodnou. Náhodné adresy jsou více využívány kvůli tomu, že nepotřebují registraci u IEEE. Náhodná adresa může být naprogramována do zařízení při výrobě a nebo vygenerována při běhu. Veřejné adresy se dále dělí na statické a privátní. Statické slouží jako náhrada za veřejnou. Mohou být vygenerovány při zapnutí a nemohou být změněny, dokud nedojde k resetování zařízení. Privátní mají podskupinu $non - resolvable$, která je náhodná a dočasná. Tato skupina se moc často nevyužívá. Druhou skupinou je $resolvable$. Je generována pomocí Identity Resolving Key (IRK) a náhodným číslem. Mění se periodicky i v době připojení, díky tomu je možnost se vyhnout sledováním neznámým zařízení. [2]

1.2.5 Host Controller Interface

HCI vrstva slouží pro komunikaci mezi vrstvou host a controller. Tyto vrstvy mohou být implementovány spolu ve stejném chipsetu a nebo odděleny. Pokud jsou odděleny, komunikace mezi nimi může probíhat pomocí tří sběrnic UART, USB a SDIO. Zprávy, které si vrstvy mezi sebou posílají jsou např. zprávy pro konfiguraci controlleru a nebo zprávy pro kontrolu a parametry spojení. [2]

1.2.6 Logical Link Control a Adaption Protocol

Tato vrstva se stará o úpravu paketů z vyšších vrstev pro vrstvy nižší, tak aby velké pakety byly rozděleny na menší pakety a vešly se do maximální možné velikosti zprávy BLE protokolu. Vrstva se stará hlavně o dva hlavní protokoly Attribute Protocol (ATT) a Security Manager Protocol (SMP). [2]

1.2.7 Role zařízení

BLE protokol má dvě hlavní role, zařízení se může chovat jako *central* nebo *peripheral*. Zařízení, které se chová jako *peripheral* vysílá advertising pakety a přijímá žádosti o připojení od dalších zařízení. Dalším zařízením může být *broadcaster*, které také vysílá advertising pakety, ale nepřijímá připojení, pouze o sobě dává vědět okolním zařízením. Takovýto druh zařízení můžeme

najít v technologii Beacon. Tato technologie se používá v marketingu, kdy se přiblížením telefonu zákazníka k Beacon zařízení zobrazí notifikace např. o aktuálních slevách. Dalším příkladem je lokalizace uvnitř budov. Zařízení s rolí *central* skenuje ostatní zařízení, které zasílají advertising pakety a může iniciovat připojení k zařízení typu *peripheral*. Role *central* navíc nabízí možnost být připojený k více zařízení najednou. Poslední rolí je *observer*, který se liší tím, že není schopný iniciovat připojení. Rozdíly mezi jednotlivými rolemi jsou uvedeny v následující tabulce. [2]

Broadcaster	Peripheral	Observer	Central
Není nutnost přijímače	Nutnost vysílače a přijímače	Není nutnost vysílače	Nutnost vysílače a přijímače
Jednosměrný přenos	Obousměrný přenos	Jednosměrný přenos	Obousměrný přenos
Sníženy požadavky na HW a SW	Vyžaduje full BLE stack	Sníženy požadavky na HW a SW	Vyžaduje full BLE stack

Tabulka 1.2: Role zařízení [2]

1.2.8 Attribute protokol - ATT

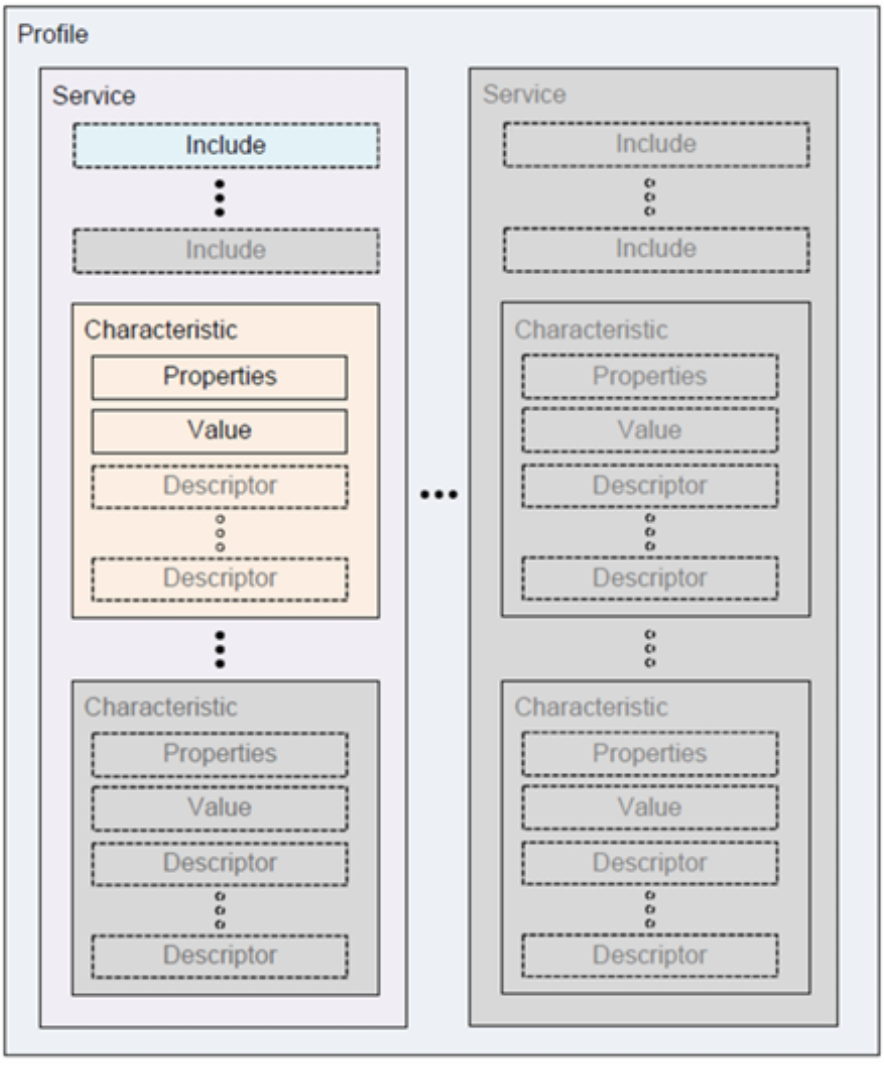
Pro komunikaci mezi mobilní aplikací a bezdrátovým mikrokontrolérem jsou pro nás důležité dva frameworky. První z nich je Generic Attribute Profile (GATT), který obsahuje protokol Attribute Protocol (ATT). Tento protokol definuje, jak jsou data pro klienta uspořádána na serveru. Server se stará o zpracování příkazů od klienta, kde může například zasílat notifikace klientovi o změně nějaké veličiny. Naproti tomu klient zasílá na server různé požadavky jako zapsání nebo čtení určité charakteristiky. Data na serveru jsou strukturována do tzv. atributů. Každý atribut má svůj unikátní identifikátor UUID, který může být 16-bitový nebo 128-bitový. Dále server přiřazuje svým atributům 16-bitové číslo - *attributehandle*. Tato hodnota slouží k tomu, aby každý atribut měl unikátní identifikátor při době připojení. Adresa může nabývat hodnot 0x0001-0xFFFF, kde hodnota 0x0000 je rezervována. Poslední vlastností atributů jsou práva ke čtení, zápisu, notifikaci a indikaci. [2]

1.2.9 Generic Attribute Profile - GATT

Framework GATT pracuje se službami a charakteristikami, tedy atributy pro které definuje vlastnosti jako možnost zápisu, čtení nebo notifikací. Od ATT se liší v tom, že role jsou určeny při výměně dat (žádost ↔ odpověď), takže se zařízení může chovat jako server nebo i jako klient. [2]

Služby

Služba vždy obsahuje jeden a nebo více atributů (charakteristik). Příkladem je standardní SIG služba *battery*, která obsahuje charakteristiku *batterylevel*. Služba obsahuje i další pomocné atributy jako *service_declarations* nebo *characteristis declarations*. Obrázek níže popisuje strukturu služby, která obsahuje charakteristiky. Každá charakteristika má své vlastnosti a hodnoty. Služba může odkazovat na další služby pro rozšíření. [2]



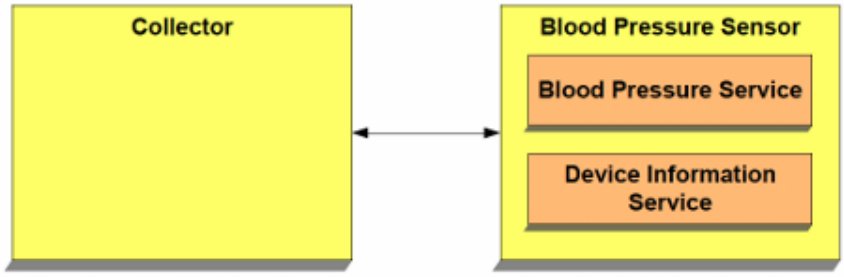
Obrázek 1.7: Struktura služby - GATT [2]

Charakteristiky

Charakteristiky mohou reprezentovat nějakou veličinu jako je stav baterie, teplota, tlak apod. Jak už bylo zmíněno výše, charakteristikách se dají přiřadit určité vlastnosti. Dále pak charakteristiky mají své deskriptory, které obsahují informace o dané veličině. [2]

Profily

Profil definuje chování mezi serverem a klientem, kde řeší jejich charakteristiky a zabezpečení. Podobně jako u služeb existují standardní SIG profily jako například $BloodPressureProfile$. Specifikace profilu obsahuje role zařízení, použití služeb a charakteristik, informace o připojení a zabezpečení. [2]



Obrázek 1.8: Profil Blood pressure [2]

Kapitola 2

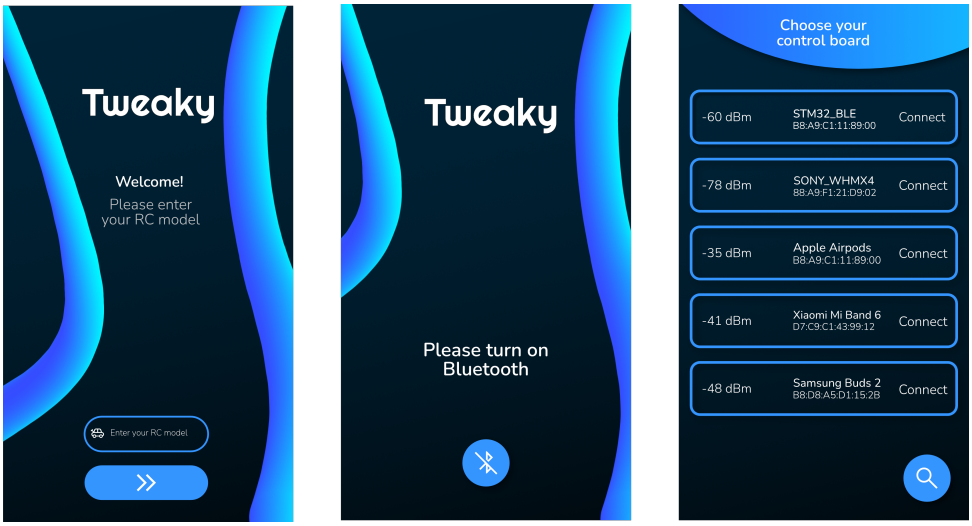
Mobilní aplikace pro ovládání RC modelu

2.1 Flutter

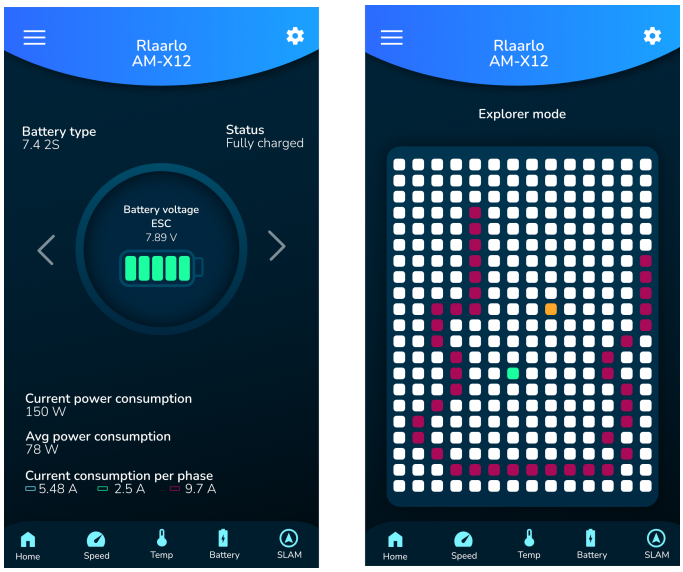
Flutter je multiplatformní framework od společnosti Google umožňující vytvářet grafické aplikace pro různá zařízení. Je určený pro systémy Android, Windows, Linux a MacOS. Výhodou je možnost mít jeden zdrojový kód, který se pak může využít pro více zařízení najednou, a tím zrychlit samotný vývoj. Nevýhodou naopak může být menší výkon oproti nativním aplikacím. Framework využívá objektově orientovaný programovací jazyk Dart, který je syntaxí velmi podobný jazyku C. Struktura kódu Flutter frameworku připomíná CSS styly nebo XAML soubory pro grafické definice. Framework dále nabízí tzv. hot reload, což umožňuje rychle aktualizovat aplikaci bez nutnosti kompilovat celou aplikaci odznova. Pro vývoj aplikace lze zvolit Android Studio a nebo Visual Studio Code, ve kterém byla aplikace napsána. Před samotným vývojem je nutné stáhnout si Flutter SDK a další pomocné nástroje. Přesné kroky jsou uvedeny na webu společnosti Googlu. Lze si stáhnout Android simulátory a vyvíjet přímo na počítači. Pokud chceme debugovat na mobilním telefonu, pak je nutné povolit USB ladění. [9]

2.2 Funkce aplikace a grafický návrh

Uživatelské rozhraní bylo navrženo v grafickém editoru Affinity Photo. Navržené rozhraní sloužilo při vývoji aplikace jako předloha a některé věci byly pak v aplikace změněny. Rozhraní obsahuje 5 karet, z nich každá poskytuje informace o měřených veličinách či chybovém hlášení.



Obrázek 2.1: Úvodní obrazovky aplikace



Obrázek 2.2: Informační karty

2.3 Popis kódu aplikace

Zdrojový kód aplikace je rozdělen na několik částí. Složka *routes* obsahuje kód každé obrazovky, ve složce *logic* je kód starající se o komunikaci aplikace s řídicí jednotkou a složka *graphics* obsahuje definice jednotlivých grafických prvků (tlačítka, text...). Aplikace používá dvě hlavní knihovny GetX a Flutter Blue Plus. První zmíněná slouží ke správě stavů widgetů. Ve Flutter frameworku se widgety dělí na *stateful* a *stateless*. Příkladem může být text, který je statický. Oproti tomu např. tlačítko pro zapnutí Wi-Fi je widget typu *stateful*. Toto je nutné v kódu ošetřovat, tímto vzniká dlouhý a nepřehledný kód. Díky frameworku je použití stavů widgetů jednoduché. Widget, který se má měnit, se přidá jako argument do funkce *Obx*(). Pokud se hodnota v nějakém textu změní, widget *Text* se automaticky překreslí. Dále jsou v knihovně pomocné metody pro přechod mezi obrazovkami a kontrolér pro správu hodnot, které se mají na obrazovce měnit. Druhou knihovnou je Flutter Blue Plus sloužící pro obsluhu Bluetooth komunikace. Její API obsahuje metody pro vyhledávaní zařízení, připojení a odpojení. Lze s ní vyhledávat služby, charakteristiky a číst z nich pomocí jejich identifikátorů. Dále lze u charakteristik nastavit naslouchání notifikací, kdy při změně charakteristiky aplikace přijme informaci o změně nebo např. nastavovat velikost MTU jednotky. V aplikaci je taktéž využita knihovna Syncfusion, která nabízí možnost vykreslení různých ukazatelů a grafů.

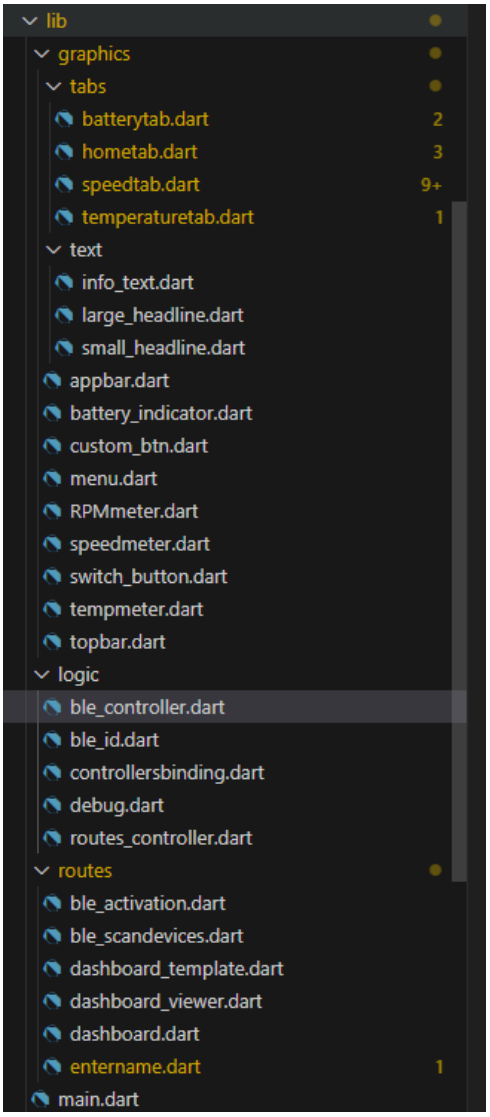
Stateful widget	Stateless widget
Mění svůj stav	Nemění svůj stav
Používá metodu CreateState() a vrací svůj stav	Používá metodu build() a vrací widget
Může se dynamicky měnit, setState() slouží pro překreslení widgetu	Zůstává stejný po dobu běhu aplikace

Tabulka 2.1: Rozdíly mezi *stateless* a *stateful* widgety[10]

Ukázka níže obsahuje vyhledávání služeb a charakteristik zařízení. V kódu se vyhledává daná služba a její charakteristiky pomocí UUID. Dále se zde nastavuje velikost MTU jednotky a povolují notifikace u jednotlivých charakteristik.

```
void discover(BluetoothDevice d) async {
    List<BluetoothService> services = await d.discoverServices()
    ;
    for (var service in services) {
      //print('Service found! rssi: ${service.uuid}');
      if (service.uuid == Guid(dashboardID)) {
        //print('Characteristic Dashboard added!');
        for (var characteristic in service.characteristics) {
          print('Characteristics Dashboard: ${characteristic.
    uuid}');
          if (characteristic.uuid == Guid(temperatureID)) {
            temperature = characteristic;
          } else if (characteristic.uuid == Guid(voltageID)) {
            voltage = characteristic;
          } else if (characteristic.uuid == Guid(IMU_ID)) {
            IMU = characteristic;
          } else if (characteristic.uuid == Guid(dutyServoID)) {
            servoPWM = characteristic;
          } else if (characteristic.uuid == Guid(dutyMotorID)) {
            motorPWM = characteristic;
          }
        }
        dashboard = service;
        listChar = service.characteristics;
        final mtu = await d.mtu.first;
        Debug.logInfo('MTU: $mtu');
        await d.requestMtu(223);
        await temperature.setNotifyValue(true);
        await IMU.setNotifyValue(true);
        await voltage.setNotifyValue(true);}
```

Ukázka kódu 2.1: Metoda *discover*()

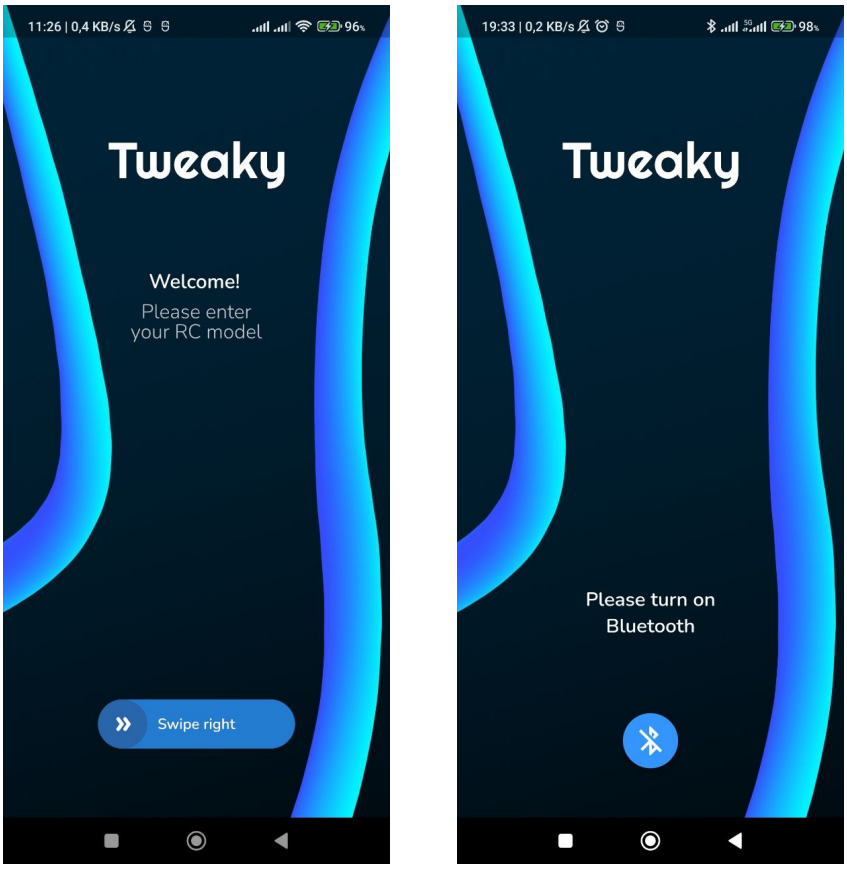


Obrázek 2.3: Rozdělení aplikace

Každá obrazovka aplikace obsahuje metodu *build(BuildContext context)*, jejíž vstup je námi vytvořená obrazovka. Obrazovka je složená z jednotlivých widgetů *Container, Scaffold, Center* a dalších. Každý widget má vstupní argumenty pomocí, kterých se vytváří grafické prvky. Dále jsou tam widgety pro polohování objektů *Center* a *Column*. Na úvodní obrazovce je tlačítko *SwipeButton* sloužící pro přechod na další obrazovku, kde v metodě *onSwipe* je zjišťováno zdali je Bluetooth zapnuté. Pokud je vypnuté, zobrazí se obrazovka s výzvou zapnutí Bluetooth. Při již zapnuté konektivitě dojde k přeskočení obrazovky na obrazovku s vyhledáváním zařízení.

```
class EnterName extends StatelessWidget {
  const EnterName({super.key});

  @override
  Widget build(BuildContext context) {
    return Container(
      constraints: const BoxConstraints.expand(),
      decoration: const BoxDecoration(
          image: DecorationImage(
        image: AssetImage("assets/images/tweaky.png"),
        fit: BoxFit.cover,
      )),
      child: Scaffold(
        resizeToAvoidBottomInset: true,
        backgroundColor: Colors.transparent,
        body: Center(
          child: Column(
            mainAxisAlignment: MainAxisAlignment.end,
            crossAxisAlignment: CrossAxisAlignment.center,
            children: <Widget>[
            Padding(
            padding: const EdgeInsets.symmetric(vertical: 70),
            child: SwipeButton.expand(
                  thumb: Icon(
                    Icons.double_arrow_rounded,
                    color: Colors.white,
                  ),
                  width: 200,
                  height: 50,
            activeThumbColor: Color.fromARGB(255, 41, 101, 170),
            activeTrackColor: Color.fromARGB(255, 35, 124, 207),
                  child: SmallHeadline(text: "Swipe right"),
                  onSwipe: () async {
                    if ((await FlutterBluePlus.isOn) == true) {
                      Get.to(() => const BleScanDevices());
                    } else {
                      Get.to(() => const BleActivation());
                    }
                  },
                ),),],),),),);}}
```

Ukázka kódu 2.2: Úvodní strana aplikace



(a) : Úvodní obrazovka aplikace **(b) :** Zapnutí Bluetooth

Obrázek 2.4: Úvodní obrazovky

Níže je kód widgetu *ListTile*, pomocí něho jsou zobrazena zařízení v okolí spolu s intenzitou přijímaného signálu RSSI v dBm. Zařízení jsou zobrazena pomocí listu *scanResultList*, do kterého jsou zařízení v okolí přidávána pomocí kontroléru. Jednotlivé výpisy jsou ohraničeny obdélníkovým okrajem s možností se připojit na dané zařízení. Pokud se zařízení připojí, dojde k přechodu na hlavní obrazovku aplikace, k tomu slouží metoda *Get.to(() => const DashboardViewer ())*.

```
    return ListTile(
       shape: RoundedRectangleBorder(
       side: const BorderSide(
             width: 2,
            color: Color.fromARGB(255, 53, 149, 255)),
            borderRadius: BorderRadius.circular(20),  ),
                 onTap: () async {
                 if (await (controller
                .connect(controller.scanResultList[index]))) {
                    controller.discover(
                    controller.scanResultList[index].device);
                    }
    //controller.connect(controller.scanResultList[index]);
                    controller.connectedDevice =
                    controller.scanResultList[index].device;
                          controller.index.value = index;
                          Get.to(() => const DashboardViewer());
                        },
                        title: Text(
                          controller
        .scanResultList[index].device.name.isNotEmpty
        ? controller.scanResultList[index].device.name
                              : 'No name device',
                          style: const TextStyle(
                              fontWeight: FontWeight.w500,
                              fontFamily: 'Nunito',
                              fontSize: 15,
            color: Color.fromARGB(255, 255, 255, 255)),
                        ),
                    subtitle: Text(
            controller.scanResultList[index].device.id.id,
                        style: const TextStyle(
                        fontWeight: FontWeight.w500,
                        fontFamily: 'Nunito',
                        fontSize: 15,
            color: Color.fromARGB(139, 255, 255, 255)),
                        ),
                    trailing: Text(
                        controller.connected.value &&
                            index == controller.index.value
                            ? 'Disconnect'
                            : 'Connect',
                        style: const TextStyle(
                            fontWeight: FontWeight.w500,
                        fontFamily: 'Nunito',
                        fontSize: 15,
                    color: Color.fromARGB(255, 255, 255, 255)),
                        ),
                    leading: Obx(
                        () => Text(
        '${controller.scanResultList[index].rssi} \ndBm',
                        style: const TextStyle(
                            fontWeight: FontWeight.w500,
                    fontFamily: 'Nunito',
fontSize: 15,color: Color.fromARGB(255, 255, 255, 255)), ), ));
```

Ukázka kódu 2.3: Vyhledávání zařízení

Obrázek 2.5: Obrazovka s vyhledáváním zařízení

2.3.1 Hlavní karty aplikace

Karta Home

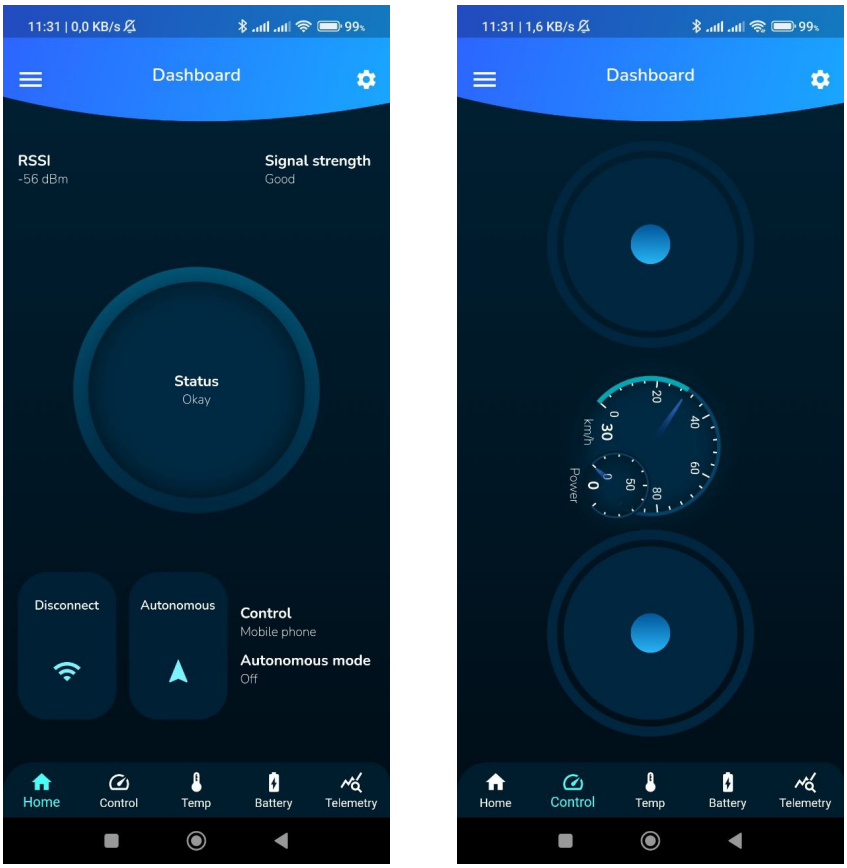
Po připojení se zobrazí karta Home, která obsahuje základní údaje o připojení a stavu systému. V horní části obrazovky je zobrazována hodnota RSSI připojeného zařízení. Uprostřed je pak ukazatel stavu měřených veličin, zdali jsou v rozmezí. Ve spodní části jsou pak tlačítko pro připojení a přepnutí se do autonomního režimu. Dolní lišta pak obsahuje seznam dalších karet, mezi kterými lze přepínat.

Karta Control

V kartě Control jsou umístěny dva joysticky, jeden pro řízení ESC regulátoru a druhý pro ovládání servomotoru. Uprostřed joysticků jsou pak ukazatele rychlosti a výkonu. Pomocí joysticků se zasílají do mikrokontroléru příkazy pro změnu šířky pulzů PWM. Pokud je joystick vychýlen, *listener*, který každých 100 ms kontroluje hodnotu pozice joysticku, předá pozici joysticku do proměnné *details*, která je dále zpracována pro zaslání příkazu. Předtím než je vyslán příkaz, dojde ke kontrole zdali se hodnota střídy už dříve neposílala, aby se omezil počet zaslaných zpráv.

```
Joystick(
     listener: (details) async {
     controller.dutyMotor.value = -details.x * 0.5 * 100;
     int intervalPWM_middle = 150;
     int commandPWM =
     intervalPWM_middle + controller.dutyMotor.value.round();
     if (prev_dutyMotor != commandPWM)
         controller.setDutyMotor(commandPWM);
         print('PWM_Motor: ${commandPWM}');
         prev_dutyMotor = commandPWM;
         },
     ),

 Joystick(
     listener: (details) async {
     controller.dutyServo.value = details.y * 0.4 * 100;
     int intervalPWM_middle = 139;
     int commandPWM =
     intervalPWM_middle + controller.dutyServo.value.round();
     if (prev_dutyServo != commandPWM)
     controller.setDutyServo(commandPWM);
     print('PWM_Servo: ${commandPWM}');
     prev_dutyServo = commandPWM;
           },
         ),
```

Ukázka kódu 2.4: Joysticky



(a) : Karta Home **(b) :** Karta Control

Obrázek 2.6: Karty aplikace

Karta Temperature a Battery

Tyto karty zobrazují hodnoty teploty a napětí akumulátorů. Pomocí šipek lze přepínat mezi jednotlivými akumulátory. Dále karty ukazují průměrné, maximální a minimální naměřené hodnoty veličin.

Karta Telemetry

Poslední karta Telemetry vykresluje do grafu průběh zrychlení v jednotlivých osách. Rozsah je nastavený na $\pm$ 8 g.

2.4 Zpracování dat ze senzorů

2.4.1 Teplota

Přijatá hrubá naměřená data z NTC termistorů se z jednotlivých bytů zpětně převádí na *integer* bitovým posunem. Tato hodnota je převedena na napětí s referenčním napětím 3.3 V a 12-bitovým rozlišením, tedy 4096 hodnot. V dalším bodě je vypočítaný aktuální odpor termistoru, který je vložen do β rovnice pro výpočet teploty. Hodnota β koeficientu činí 3947 K pro rozsah teplot od 0°C do 100°C a byla zjištěna z webu výrobce Vishay. [12] Tato rovnice slouží jako aproximace závislosti odporu na teplotě termistoru, která je nelineární.

$$R(T) = R_{T_0} e^{\beta(\frac{1}{T_0} - \frac{1}{T})} \tag{2.1}$$

$$T_0 = 285.15\ K \tag{2.2}$$

$$R_{T_0} = 10\ k\Omega,\ odpor\ termistoru\ při\ 25°C \tag{2.3}$$

$$\beta = 3947K \tag{2.4}$$

$$T = aktuální\ teplota\ termistoru \tag{2.5}$$

Teplota T_0 je 25°C a R_{T_0} je odpor termistoru při této teplotě. Po získání teploty je dále zjišťována minimální, maximální a průměrná hodnota. Na další straně je uveden kód pro získání teploty.

```
temperature.onValueChangedStream.listen((value) async {
       int temp1raw = (value[1] << 8) + value[0];
       int temp2raw = (value[3] << 8) + value[2];
       temperature1.value = (3.3 / 4096) * temp1raw;
       temperature2.value = (3.3 / 4096) * temp2raw;
       double resistance1 =
      ((10000) * (3.3 - temperature1.value)) / temperature1.
value;
       double resistance2 =
      ((10000) * (3.3 - temperature2.value)) / temperature2.
value;
       int beta = 3947;
       int R0 = 10000;
       temperature1.value =
       beta / (log(resistance1 / 10000) + (beta / 298.15)) -
273.15;
       temperature2.value =
       beta / (log(resistance2 / 10000) + (beta / 298.15)) -
273.15;
       if (temperature1MAX.value < temperature1.value)
         temperature1MAX.value = temperature1.value;

       if (temperature2MAX.value < temperature2.value)
         temperature2MAX.value = temperature2.value;

       if (temperature1MIN.value > temperature1.value)
         temperature1MIN.value = temperature1.value;

       if (temperature2MIN.value > temperature2.value)
         temperature2MIN.value = temperature2.value;
       temperature1List.add(temperature1.value);
       double sum1 = 0;
       if (temperature1List.length == 5) {
         for (var temp in temperature1List) {
           sum1 = sum1 + temp;
         }
         temperature1AVG.value = sum1 / 5;
         temperature1AVG.value.toPrecision(2);
         temperature1List.clear();
       }
       temperature2List.add(temperature2.value);
       double sum2 = 0;
       if (temperature2List.length == 5) {
         for (var temp in temperature2List) {
           sum2 = sum2 + temp;
         }
         temperature2AVG.value = sum2 / 5;
         temperature2AVG.value.toPrecision(2);
         temperature2List.clear();
       }
     }
```

Ukázka kódu 2.5: Teplota

2.4.2 Napětí

Podobně jako u teploty jsou byty převedeny na *integer* a pomocí rovnice napěťového děliče je získána hodnota napětí na akumulátorech. Z webu Ampow byla použita data, kde pomocí webu curve.fit byla data proložena přímkou s koeficienty $a = 1.272e^{-2}$ a $b = 7.068$ pro odhad stavu kapacity akumulátorů. [14] [13]

```
voltage.onValueChangedStream.listen((value) {
int volt1raw = (value[1] << 8) + value[0];
int volt2raw = (value[3] << 8) + value[2];

voltage1.value = (((3.3 / 4096) * volt1raw) * (3300 + 33000)) /
    3300;
voltage2.value = (((3.3 / 4096) * volt2raw) * (3300 + 33000)) /
    3300;

voltage1capacity.value = (voltage1.value - 7.068e+00) / 1.272e
    -02;
voltage2capacity.value = (voltage2.value - 7.068e+00) / 1.272e
    -02;

if (voltage1capacity.value > 100) voltage1capacity.value = 100;
if (voltage2capacity.value > 100) voltage2capacity.value = 100;
if (voltage1capacity.value < 0) voltage1capacity.value = 0;
if (voltage2capacity.value < 0) voltage2capacity.value = 0;
        }
```

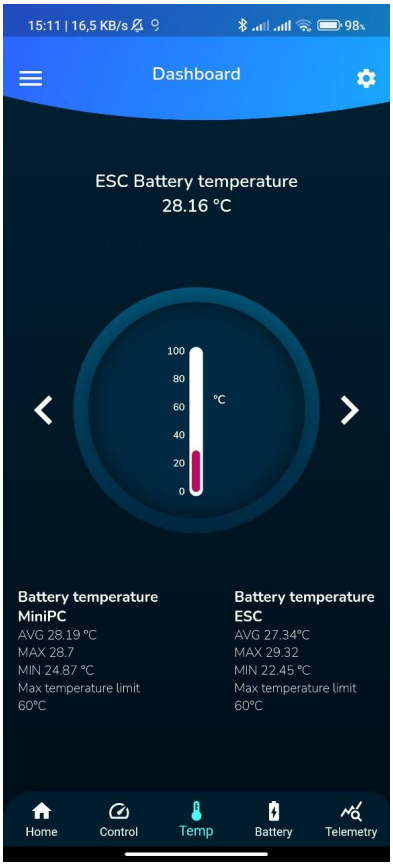
Ukázka kódu 2.6: Napětí

2.4.3 Akcelerometr

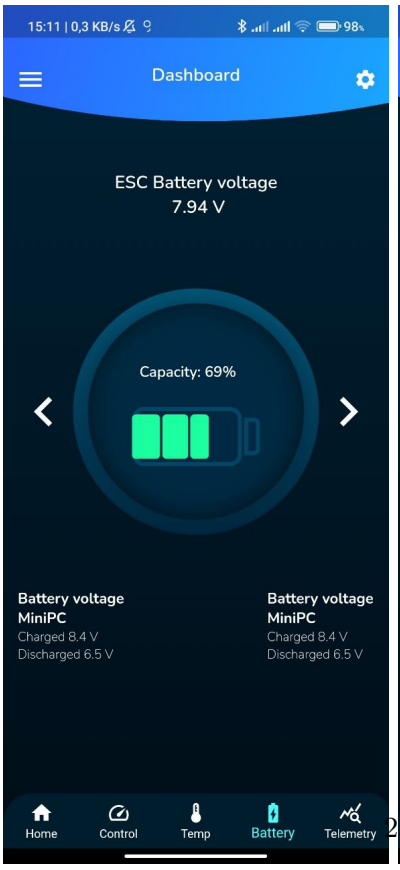
Data pro akcelerometr jsou vynásobena citlivostí 0.244 a dále převedena z mg na g (tíhové zrychlení).

```
IMU.onValueChangedStream.listen((value) {
          int acXraw = (value[1] << 8) + value[0];
          int acYraw = (value[3] << 8) + value[2];
          int acZraw = (value[5] << 8) + value[4];

          accelX = (acXraw * 0.244) / 1000;
          accelY = (acYraw * 0.244) / 1000;
          accelZ = (acZraw * 0.244) / 1000;

          accelXView.value = accelX;
          accelYView.value = accelY;
          accelZView.value = accelZ;
        }
```

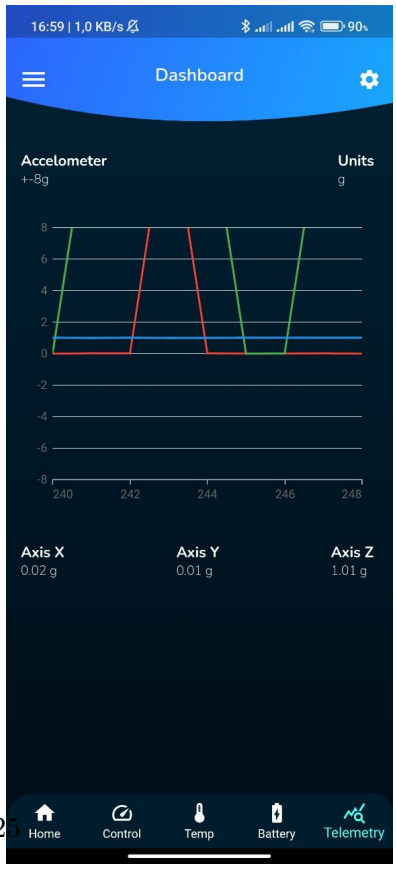
Ukázka kódu 2.7: Akcelerometr



(a) : Karta Temperature



(b) : Karta Battery



(c) : Karta Telemetry

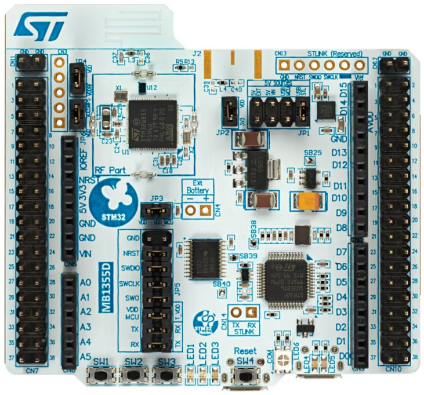
Obrázek 2.7: Karty aplikace

Kapitola 3

Návrh řídicí jednotky s BLE konektivitou

3.1 Schéma zapojení

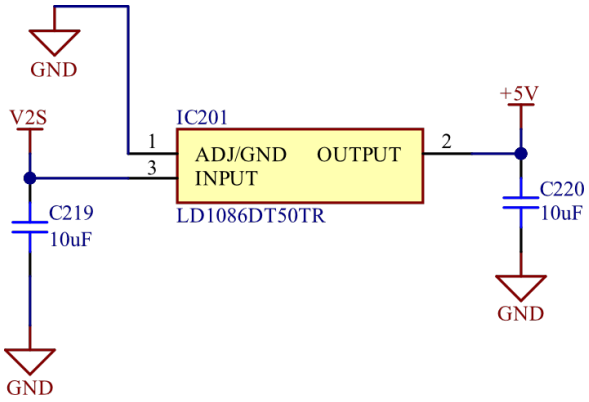
Pro modul byl zvolen mikrokontrolér STM32WB55RG s bezdrátovou komunikací, jelikož vlastním vývojový kit NUCLEO-WB55RG, na kterém jsem si odzkoušel základní komunikaci s mobilní aplikací. Čip obsahuje dvě jádra Arm Cortex-M4 64 MHz a Arm Cortex-M0+ 32 Mhz sloužící pro obsluhu bezdrátových protokolů ZigBee, BLE nebo Thread . Dále obsahuje periferie UART, I2C, SPI, USB a AD/DA převodníky. Při tvorbě schématu se vycházelo z vývojového kitu, z aplikačních poznámek a datasheetů výrobce. Deska byla navržena v softwaru Altium Designer, jedná se o program sloužící pro návrh desek plošných spojů a simulaci obvodů.



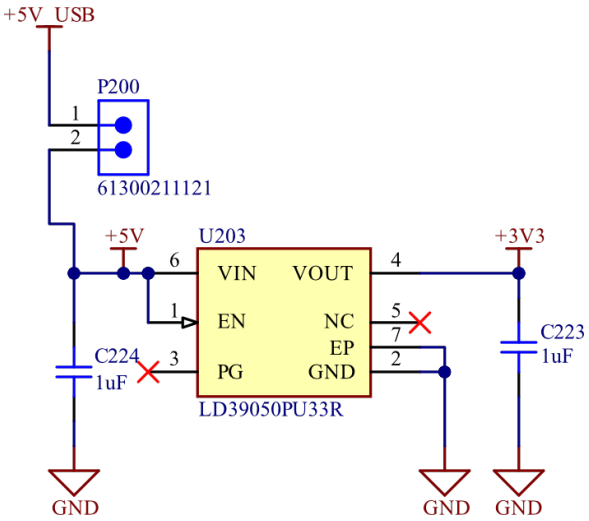
Obrázek 3.1: Vývojový kit NUCLEO-WB55RG [7]

3.1.1 Napájení

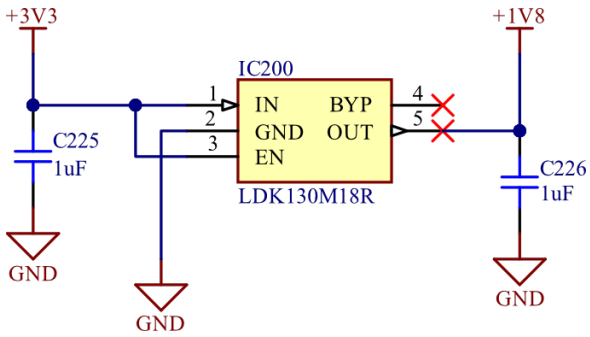
Měřicí deska je napájena z 2S Li-Pol akumulátoru s nominálním napětí 7.4 V, které je dále sníženo LDO regulátorem na 5 V. V schématu jsou poté další 3 regulátory pro napájecí napětí 3.3 V mikrokontroléru, akcelerometru 1.8 V a připojení externího senzoru napájeného 5 V. Regulátory LDK130M18R a LD39050PU33R mají vstupní a výstupní stabilizační kondenzátory o velikostech 1 μF, které jsou uvedeny v datasheetu. Pro 5V regulátory LD1086DT50TR je hodnota kondenzátorů 10 μF.



Obrázek 3.2: LDO regulátor 5 V



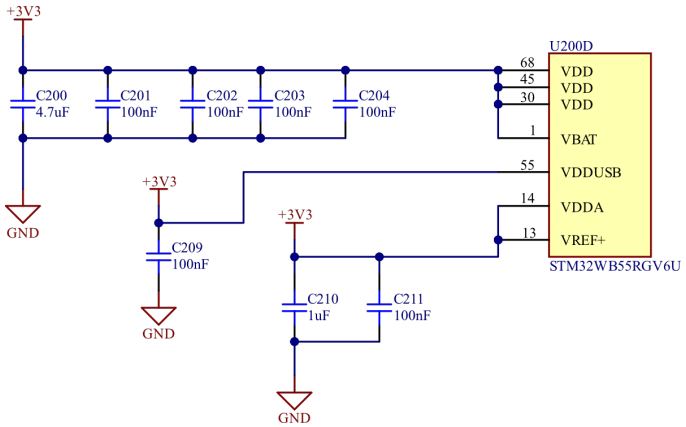
Obrázek 3.3: LDO regulátor 3.3 V



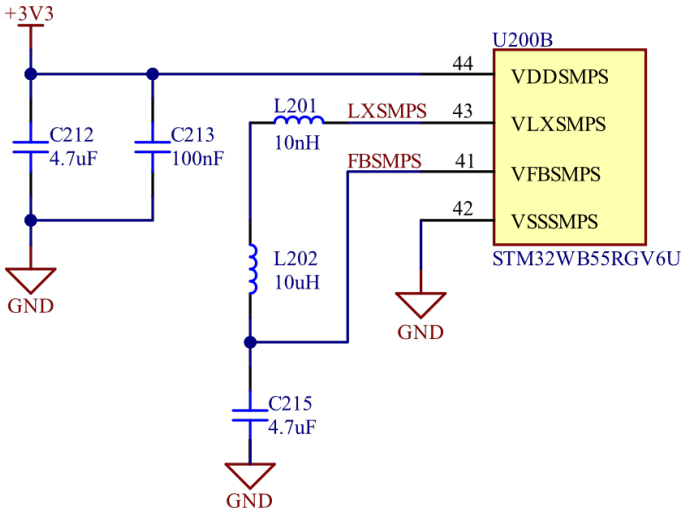
Obrázek 3.4: LDO regulátor 1.8 V

3.1.2 Piny pro napájení

Ke každému napájecímu pinu je zapojen blokovací kondenzátor o hodnotě 100 nF doporučené výrobcem v datasheetu.

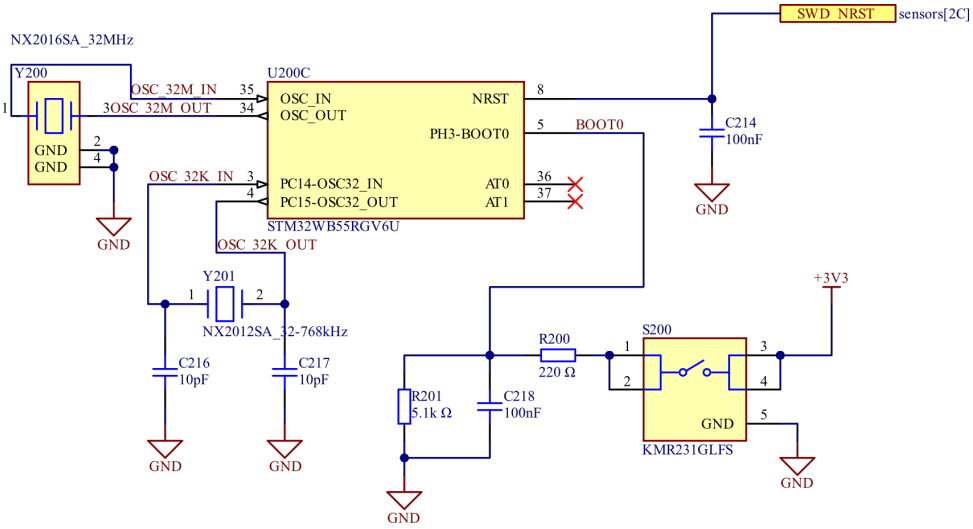


Obrázek 3.5: Hodnoty blokovacích kondenzátorů pro napájecí piny [8]



Obrázek 3.6: Zapojení zpětné vazby pro interní SMPS [8]

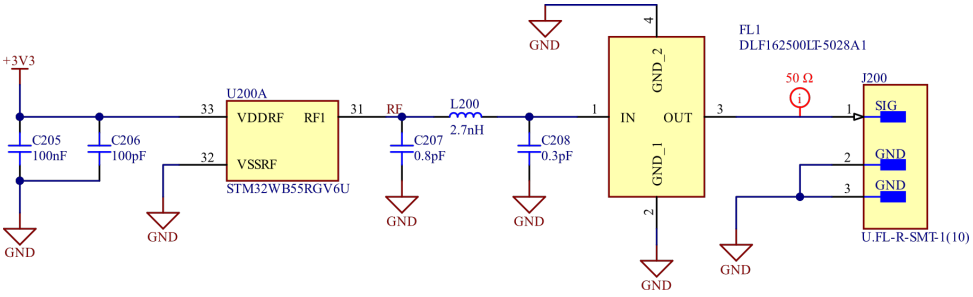
Schéma níže obsahuje dva externí krystalové oscilátory o hodnotách 32 MHz a 32.768 kHz. Dále je zde blokovací 100nF kondenzátor pro resetovací pin a tlačítko pro *BOOT*0 pin sloužící pro výběr, ze které paměti poběží program.



Obrázek 3.7: Zapojení externích krystalových oscilátorů [8]

3.1.3 RF část

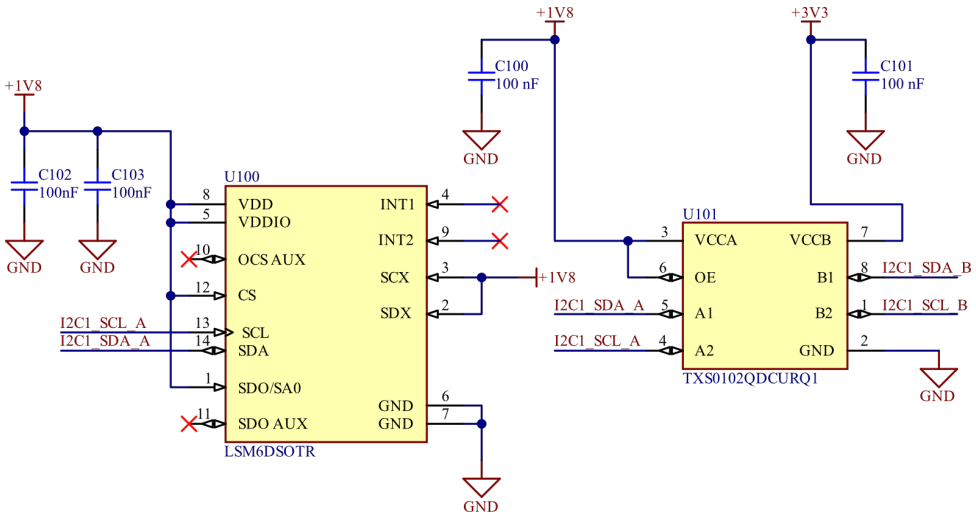
K pinu VDDRF (napájení pro blok bezdrátové komunikace) jsou připojeny dva blokovací kondezátory o velikostech 100 nF a 100 pF. Jelikož impedance výstupního pinu RF není přesně 50 Ω je nutné připojit k pinu člen pro impedanční přizpůsobení, aby nedocházelo k odrazům a ztrátě výkonu. Hodnoty cívky a kondenzátorů jsou doporučeny přímo výrobcem. Před výstupní 50Ω U.FL konektor je předřazen filtr dolní propusti pro filtrace vyšších harmonický a neharmonických signálů.



Obrázek 3.8: Schéma RF části [8]

3.1.4 IMU senzor

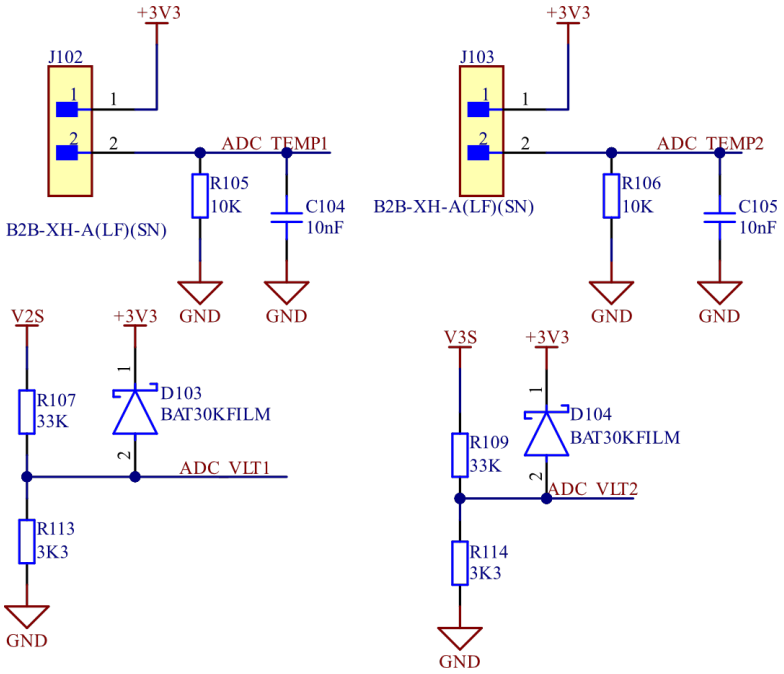
Senzor LSM6DSO byl zvolen jako inerciální jednotka. Obsahuje v sobě akcelerometr, gyroskop a teploměr. Se senzorem je možné komunikovat pomocí sběrnic I2C, SPI nebo MIPI I3C. Možné rozsahy měření pro akcelerometr jsou ±2/±4/±8/±16 g a pro gyroskop ±125/±250/±500/±1000/±2000 dps. Typické napájecí napětí je 1.8 V a kvůli tomu je mezi senzor a mikrokontrolér umístěn převodník úrovní TXS0102QDCURQ1 1.8 V ↔ 3.3 V. Pro komunikaci byla zvolena sběrnice I2C. Schéma neobsahuje potřebné pullup rezistory pro komunikaci, protože použitý převodník úrovní má vnitřní 10kΩ rezistory. Dále jsou napájecí piny opatřeny blokovacími kondezátory o hodnotě 100 nF. Pro výběr I2C komunikace je pin CS připojen k napájecímu napětí.



Obrázek 3.9: IMU senzor

3.1.5 Snímání teploty a napětí

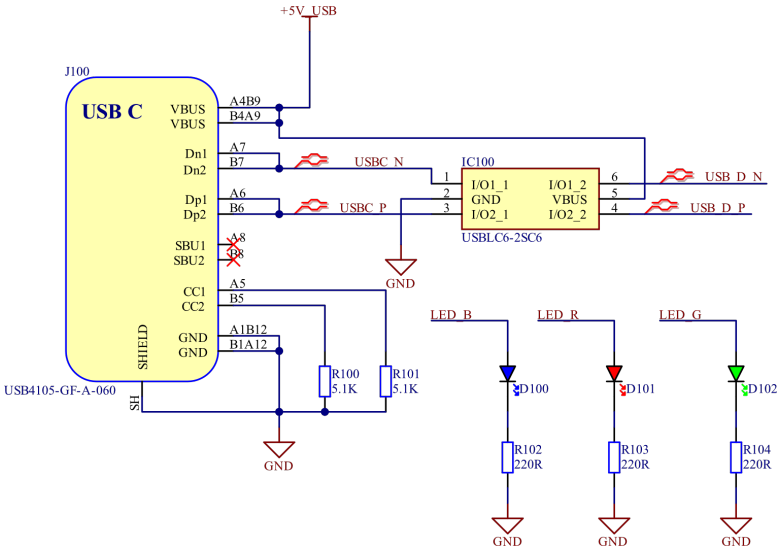
Snímání teploty probíhá za pomocí NTC termistoru o hodnotě 10 kΩ, který je připojen k napěťovému děliči. Napěťový dělič je opatřen 10 nF kondenzátorem pro stabilizaci měřeného napětí. Napětí je snímáno taktéž pomocí napěťového děliče o hodnotách 3.3kΩ a 33kΩ. Dále je připojena Schottkyho dioda pro ochranu vstupu ADC převodníku.



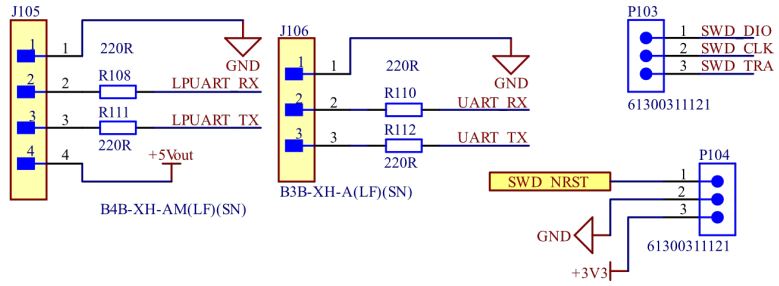
Obrázek 3.10: Snímání teploty a napětí

3.1.6 Konektivita

Pro komunikaci s mini PC jsou vyvedeny z mikrokontroléru RX a TX signály pro komunikaci přes UART rozhraní. Dále je možné komunikovat za pomocí USB. Deska obsahuje USB konektor typu C, za kterým je připojena ESD ochrana. Dále jsou vyvedeny piny pro PWM modulaci a SWD piny pro naprogramování mikrokontroléru. Pro stavy komunikace deska obsahuje 3 indikační LED.



(a) : USB C konektor a indikační LED



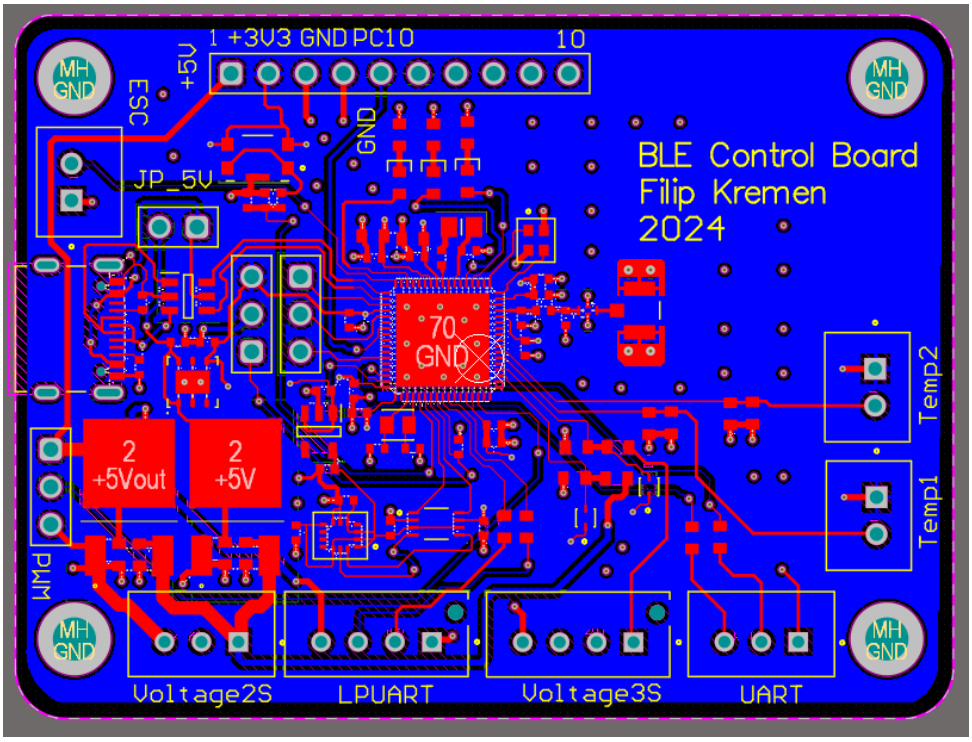
(b) : UART a SWD rozhraní

Obrázek 3.11: Konektivita

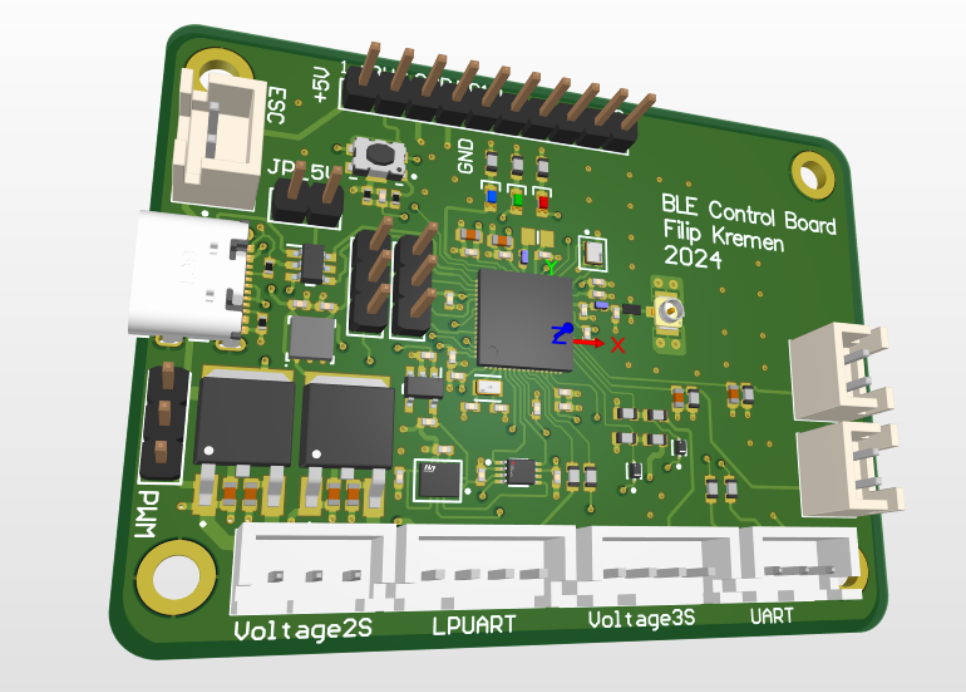
3.2 Deska plošných spojů

Pro měřicí modul byla zvolena čtyřvrstvá deska, kde horní a dolní vrstva slouží pro signály a napájení a dvě vnitřní jsou určeny pro zem. Dolní vrstva je určena pro rozvod napájecího napětí 3.3 V. Deska má tloušťku 1.6 mm, horní a dolní vrstva má hmotnost 1 oz, vnitřní pak 0.5 oz. Deska byla vyrobena ve společnosti JLCPCB. Pro dielektrikum byl zvolen materiál 7628 s relativní permitivitou $\epsilon_r = 4.4$. Jelikož návrh obsahuje RF část a USB konektivitu, je nutné dodržet požadovanou impedanci. Impedance 50 Ω pro mikropáskový spoj a daný stack-up 7628 byla pomocí kalkulátoru na webu výrobce doporučena šířka spoje 0.349 mm. Pro diferenciální pár 90 Ω se vzdáleností spojů od sebe 0.203 mm vyšla šířka spoje přibližně 0.27 mm. Rozměry šířky spojů a vzdálenosti odpovídaly hodnotám stack manageru v Altium Designer. Blokovací kondenzátory a krystalové oscilátory jsou umístěny blízko pinům pro snížení parazitní indukčnosti. Spoje pro USB jsou vedeny jako diferenciální pár. V okolí RF pinu jsou pouze kondenzátory, aby se snížilo

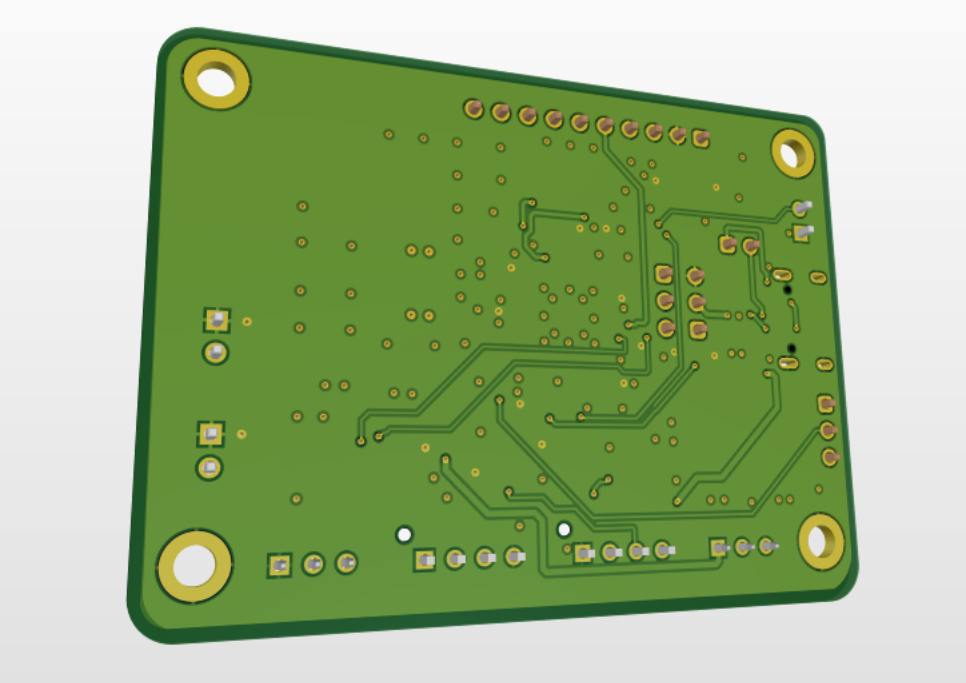
rušení do okolních obvodů. Dále jsou po desce rozmístěny via otvory pro snížení indukčnosti a zkrácení cest proudu. Na rozích desky jsou připevňovací otvory, které jsou spojeny se zemí.



Obrázek 3.12: PCB layout měřicího modulu



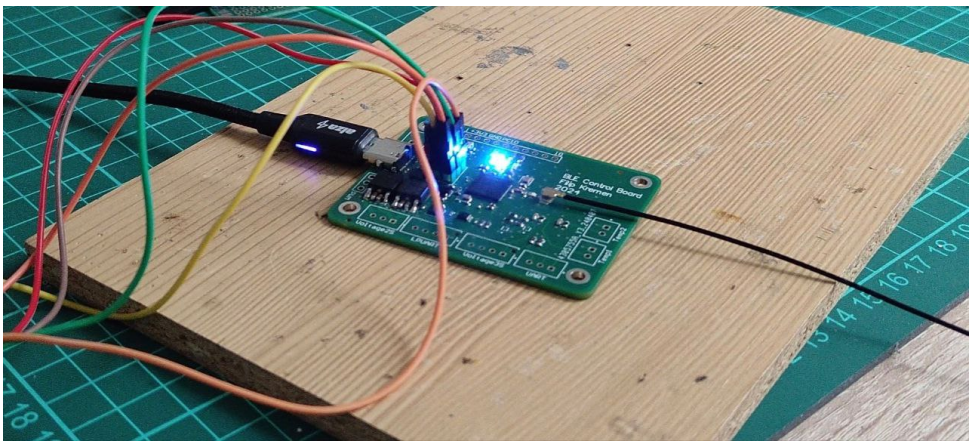
Obrázek 3.13: 3D model měřicího modulu - horní strana



Obrázek 3.14: 3D model měřicího modulu - spodní strana

3.3 Osazení a testování desky

Vyrobená deska byla osazena ručně za pomocí mikropáječky a horkovzdušné pistole, kromě mikrokontroléru, 3V3 LDO regulátoru a senzoru LSM6DSO, který byl osazen při výrobě. Po připájení součástek bylo provedeno základní měření součástek, kde se například u blokovacích kondenzátorů zjistilo, že některé nejsou připájeny korektně a to díky tomu, že na nich bylo naměřena jejich kapacita a ne celková paralelní kombinace kondenzátorů. Dále byla deska zkontrolována vizuálně pomocí kamery na telefonu. Po kontrole byla deska připojena k napájení pomocí USB kabelu a připojena k programátoru ST-Link V3, kde se deska úspěšně připojila k programátoru. K modulu je připojena všesměrová anténa se ziskem 5 dBi v pásmu 2.4-2.5 GHz.



Obrázek 3.15: Osazená deska

Value	Designator	Footprint
100nF	C100, C101, C102, C103, C201, C202, C203,	CAP_0402
	C204, C205, C209, C211, C213, C214, C218	
10nF	C104, C105	CAP_0603
4.7uF	C200, C212	CAP_0603
100pF	C206	CAP_0402
0.8pF	C207	CAP_0402
0.3pF	C208	CAP_0402
1uF	C210, C223, C224, C225, C226	CAP_0402
4.7uF	C215	CAP_0402
10pF	C216, C217	CAP_0402
10uF	C219, C220, C221, C222	CAP_0603
BLUE	D100	LED_0603_BLUE
RED	D101	LED_0603_RED
GREEN	D102	LED_0603_GREEN
BAT30KFILM	D103, D104	FP-SOD-523-IPC_C
DLF162500LT-5028A1	FL1	DLF162500LT5028A1
USBLC6-2SC6	IC100	SOT95P280X145-6N
LDK130M18R	IC200	SOT95P280X145-5N
USB4105-GF-A-060	J100	USB4105-GF-A-060
B2B-XH-A(LF)(SN)	J102, J103, J104	FP-B2B-XH-A_LF_SN-MFG
B4B-XH-AM(LF)(SN)	J105, J201	FP-B4B-XH-AM_LF_SN-MFG
B3B-XH-A(LF)(SN)	J106, J202	FP-B3B-XH-A_LF_SN-MFG
U.FL-R-SMT-1(10)	J200	FP-U_FL-R-SMT-1_10-MFG
2.7nH	L200	IND_0402
10nH	L201	IND_0402
10uH	L202	INDC2012X145N
61300311121	P101, P103, P104	61300311121
61301011121	P102	61301011121
61300211121	P200	61300211121
5.1K	R100, R101	RES_0402
220R	R102, R103, R104, R108, R110, R111, R112	RES_0603
10K	R105, R106	RES_0603
33K	R107, R109	RES_0603
3K3	R113, R114	RES_0603
220R	R200	RES_0402
5.1k	R201	RES_0402
120R	R202, R204	RES_0402
360R	R203, R205	RES_0402
KMR231GLFS	S200	FP-KMR231GLFS-MFG
LSM6DSOTR	U100	FP-LGA-14L-DM00249496-MFG
TXS0102QDCURQ1	U101	FP-DCU0008A-MFG
STM32WB55RGV6U	U200	VFQFPN68
LD1086PUR	U201, U202	SON80P400X400X100-9N-D
LD39050PU33R	U203	DFN6D_L

Tabulka 3.1: Seznam použitých součástek

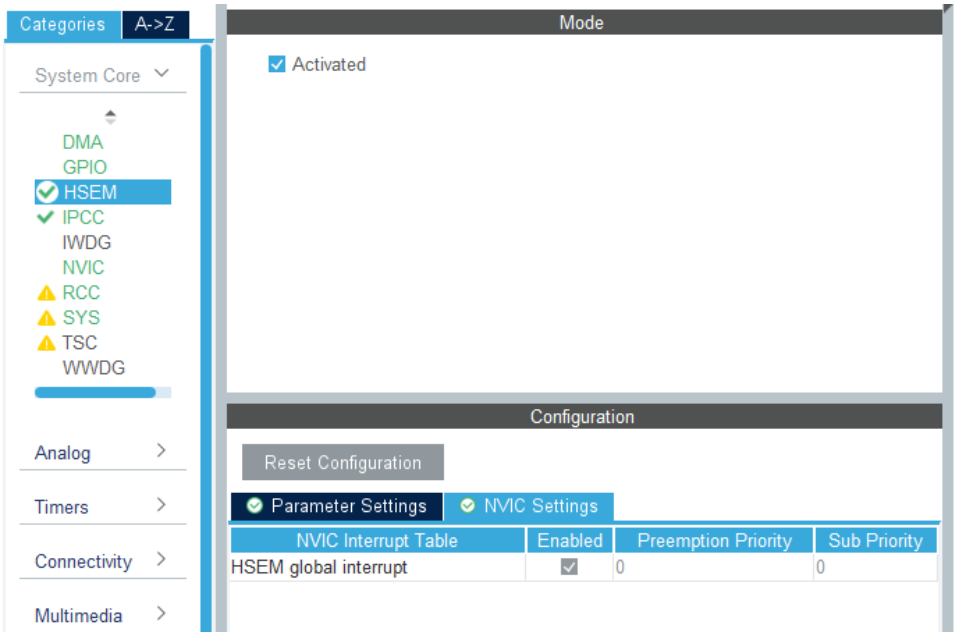
Kapitola 4

Firmware pro měřicí modul

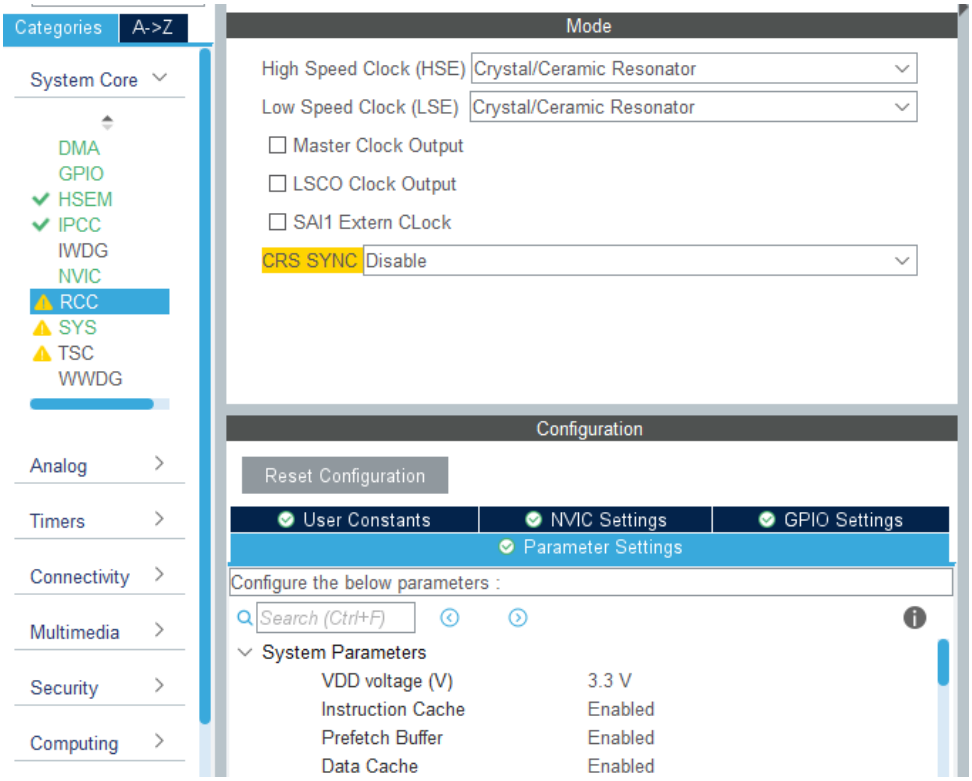
Kód pro mikrokontrolér byl napsán ve vývojovém prostředí STM32 CubeIDE. Mikrokontrolér byl nastaven pomocí programu STM32CubeMX, který slouží pro konfiguraci pinů, periferií, knihoven a generování kódu. Předtím než je možné používat BLE konektivitu, je nutné provést pár konfiguračních kroků. Mikrokontroléry řady STM32WB disponují softwarem FUS (Firmware Upgrade Services) pro instalaci, mazání a aktualizaci bezdrátových stacků. K tomu slouží doprovodný software STM32CubeProgrammer umožňující nahrávat uživatelské programy, mazat FLASH paměť, číst z registrů apod.. Po připojení je nutné nahrát nejnovější FUS na adresu 0x080EC000. Dále je pak nutné nahrát do FLASH paměti BLE stack, konkrétně pro STM32WBRG55 je BLE Stack full určen na adresu 0x080CE000. Poté jsem si vytvořil nový projekt pro STM32WBRG55V6, kde jsem ověřil základní funkčnost pomocí rozsvícení LED a ověřil i funkcionalitu externích krystalových oscilátorů a BLE konektivity. Po lehké úpravě původní kódu pro vývojový kit, jsem nahrál kód do osazené desky, kde taktéž funguje. Níže popíšu kroky při vytváření firmwaru.

4.1 BLE konektivita

Předtím než je možné používat knihovnu STM32WPAN, je nutné zapnout několik periferií. Je nutné aktivovat HSEM a IPCC, kde HSEM je hardwarový semafor sloužící pro synchronizaci mezi sdíleními prostředky a IPCC je kontrolér pro komunikaci mezi jádry. Jako zdroje hodin jsou nastaveny dva externí krystalové oscilátory v komponentě RCC. V položce RTC je nutné aktivovat hodiny a povolit wake-up interrupt. Jako poslední věc se aktivuje RF pin. Po těchto krocích je možné aktivovat položku STM32WPAN.

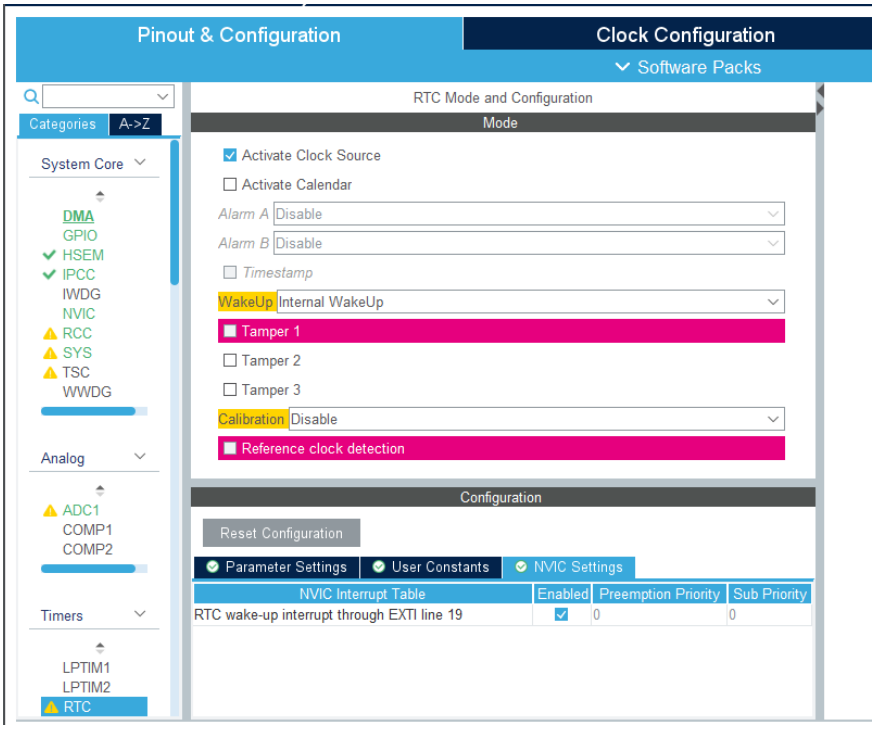


Obrázek 4.1: Aktivace HSEM a IPCC

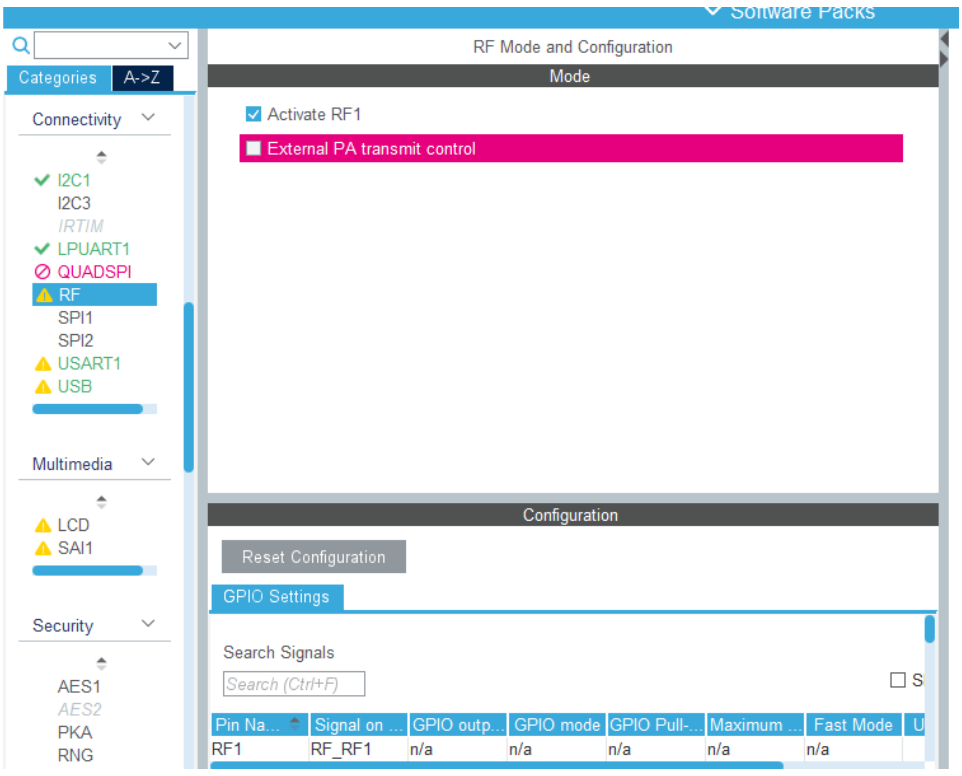


Obrázek 4.2: Externí krystalové oscilátor jako zdroje hodin

Jako typ aplikace je nastaven server profile, který má nastaven mód custom template, který nám umožní vytvářet vlastní služby a charakteristiky.

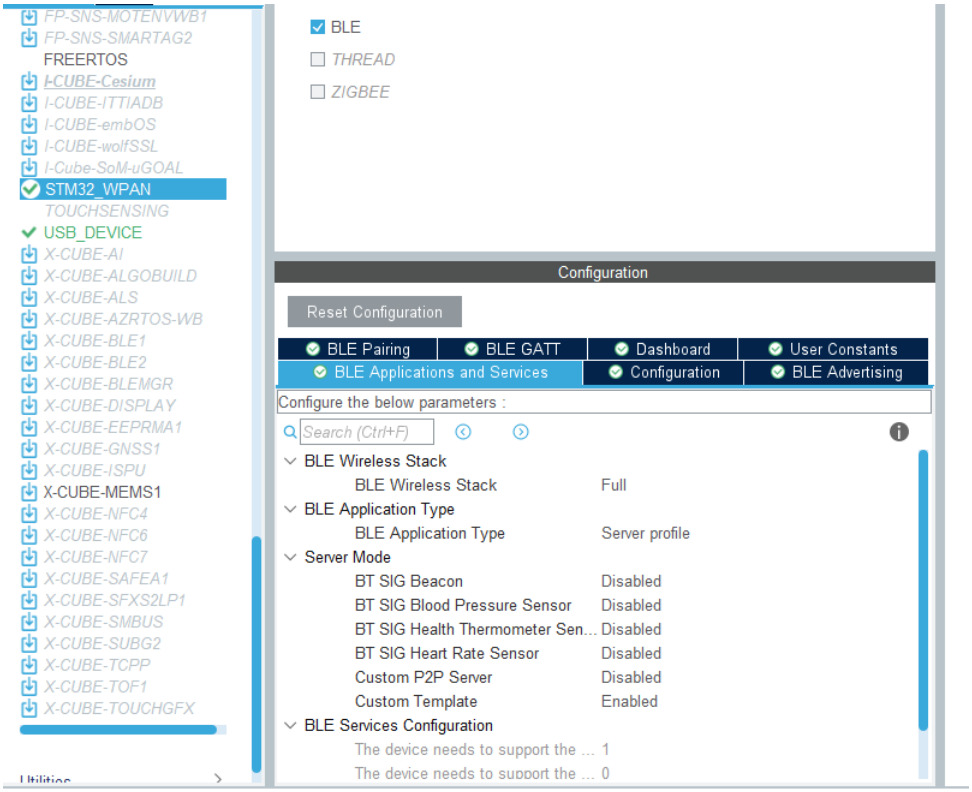


Obrázek 4.3: Interní WakeUp interrupt



Obrázek 4.4: Aktivace pinu RF

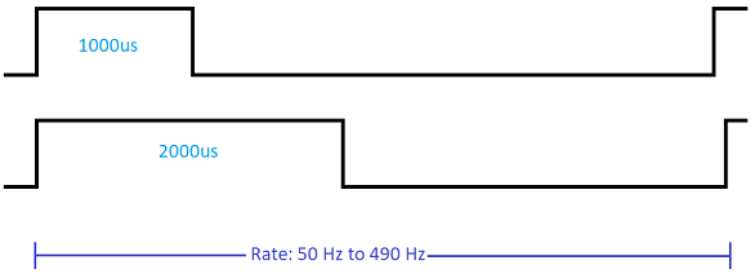
Vytvořená aplikace má jednu službu spolu s pěti dalšími charakteristikami. Jednotlivé charakteristiky jsou určeny pro měření napětí a teploty, pro IMU senzor a dále pak pro nastavení PWM modulace. Službám a charakteristikách lze přiřadit název. U charakteristiky lze nastavit její délka a identifikační číslo UUID. Dále pak zdali lze do charakteristiky zapisovat, číst či zasílat notifikace. Příklad nastavení charakteristiky je možné vidět níže. V položce Configuration lze také nastavit vysílací výkon a to až na 6 dBm.



Obrázek 4.5: BLE konfigurace

4.2 PWM modulace

Frekvence PWM modulace pro modelářské servomotory a ESC regulátory se pohybuje mezi 50 Hz a 490 Hz se šířkou pulzů od 1000 μs do 2000 μs. Požadovaná frekvence a šířky pulzu byly zjištěny na osciloskopu z původního bezdrátového ovladače. Naměřená frekvence byla 66.67 Hz. Šířka pulzů pro servomotor se pohybuje od 1000 μs do 1800 μs a pro ESC od 1000 μs do 2000 μs. Pro modulaci jsou využívány dva časovače - TIM1 a TIM2 o frekvenci 32 MHz, která je dále vydělena hodnotou 320. ARR register je nastaven na hodnotu 1499 pro požadovanou frekvenci modulace. Ve funkci *Custom_STM_App_Notification*() je možné psát vlastní kód, který se vykoná po nějaké události. Níže je uvedený kód, který po zápisu do charakteristiky modulace uloží hodnotu pro nastavení šířky pulzů do registrů CCR časovačů TIM1 a TIM2. Počáteční hodnoty šířky pulzů jsou nastaveny na 1400 μs pro servomotor a 1500 μs pro ESC.



Obrázek 4.6: Typické hodnoty frekvence a šířky pulzů PWM [6]

Obrázek 4.7: Naměřená frekvence pulzů

```
case CUSTOM_STM_DUTYSERVO_WRITE_NO_RESP_EVT:
      /* USER CODE BEGIN CUSTOM_STM_DUTYSERVO_WRITE_NO_RESP_EVT
   */
      TIM2->CCR1 = pNotification->DataTransfered.pPayload[0];
      /* USER CODE END CUSTOM_STM_DUTYSERVO_WRITE_NO_RESP_EVT */
      break;

case CUSTOM_STM_DUTYMOTOR_WRITE_NO_RESP_EVT:
      /* USER CODE BEGIN CUSTOM_STM_DUTYMOTOR_WRITE_NO_RESP_EVT
   */
      TIM1->CCR3 = pNotification->DataTransfered.pPayload[0];
      /* USER CODE END CUSTOM_STM_DUTYMOTOR_WRITE_NO_RESP_EVT */
      break;
```

Ukázka kódu 4.1: Funkce Custom_STM_App_Notification()

4.3 Nastavení USB komunikace

STM32CubeMX umožňuje využít knihovnu pro USB, a tak lze s počítačem komunikovat pomocí virtuálního COM portu. V kolonce USB je nutné zaklknout Device (FS) a poté zvolit v položce USB_DEVICE třídu Virtual Port Com. Po vygenerování kódu je třeba ještě přidat jeden řádek $LL\_HSEM\_1StepLock(HSEM, 5)$ do funkce $PeriphCommonClock\_Config(void)$, který zamezí CPU2 vypnutí HSI oscilátoru, jinak zařízení nebude rozpoznáno po připojení do PC. Po vygenerování je možné komunikovat pomocí funkcí CDC_Transmit_FS() a CDC_Receive_FS().

4.4 Komunikace s mobilní aplikací

Po vygenerování kódu pomocí STM32CubeMX se vytvoří několik souborů, v souboru *custom_app.c* se pak může implementovat samotná aplikace. V souboru *main.c* se volají funkce pro konfiguraci hodin, periferií, BLE stacku a IMU senzoru. Aplikace využívá tzv. sequenceru, což je plánovač úloh a chová se podobně jako cyklus *while*(). Jeho použití dovoluje předejít kolizím mezi úlohami. Jednotlivé úlohy se musí před použitím zaregistrovat a dále se jim musí přiřadit ID. Je možné zaregistrovat až 32 úloh, přičemž se dá každá úloha pozastavit a zase spustit. Ve funkci Custom_APP_Init() jsou jednotlivé úlohy, které měří napětí, teplotu nebo komunikují se senzorem inicializovány. Kód níže obsahuje inicializace jednotlivých proměnných pro měřené veličiny. Funkce HAL_ADCEx_Calibration_Start() slouží pro kalibraci AD převodníku, která je následována funkcí HAL_ADC_Start_DMA() spouštějící AD převodník v režimu DMA pro skenování jednotlivých kanálů. Funkce UTIL_SEQ_RegTask slouží k registraci úloh pro měření. Každé

měření je přiřazeno k hardwarovému time serveru, kde každý má určitou periodu spouštění. [5]

```
void Custom_APP_Init(void)
{
  data.Update_timer_ID = 0;
  data.Vltg_timer_ID = 1;
  data.Tmp_timer_ID = 2;
  data.IMU_timer_ID = 3;
  HAL_ADCEx_Calibration_Start(&hadc1,ADC_SINGLE_ENDED);
  HAL_ADC_Start_DMA(&hadc1, (uint32_t*)Raw, 2);
  UTIL_SEQ_RegTask( 1<< CFG_TASK_VALUE_MEASURE_ID, UTIL_SEQ_RFU,
        Dashboard_Measurement);
  HW_TS_Create(CFG_TIM_PROC_ID_ISR, &data.Update_timer_ID,
    hw_ts_Repeated, Dashboard_Measurement_Update);
  HW_TS_Start(data.Update_timer_ID, (1000000/CFG_TS_TICK_VAL)/5)
    ;
  HW_TS_Create(CFG_TIM_VLTG, &data.Vltg_timer_ID, hw_ts_Repeated
    , Dashboard_Measurement_Vltg);
  HW_TS_Start(data.Vltg_timer_ID, (1000000/CFG_TS_TICK_VAL));
  HW_TS_Create(CFG_TIM_TMP, &data.Tmp_timer_ID, hw_ts_Repeated,
    Dashboard_Measurement_Tmp);
  HW_TS_Start(data.Tmp_timer_ID, (1000000/CFG_TS_TICK_VAL));
  HW_TS_Create(CFG_TIM_IMU, &data.IMU_timer_ID, hw_ts_Repeated,
    Dashboard_Measurement_Speed);
  HW_TS_Start(data.IMU_timer_ID, (1000000/CFG_TS_TICK_VAL)/5);
  return;
}
```

Ukázka kódu 4.2: Funkce Custom_APP_Init()

4.4.1 Měření a zasílání veličin

Jako již bylo výše zmíněno pro každou veličinu je nastavený časovač pomocí, kterého se periodicky získává daná veličina. Pro měření napětí a teploty je nastaven 12-bitový AD převodník v DMA režimu, což znamená, že se naměřená data ukládají do paměti bez přítomnosti CPU a díky tomu může procesor provádět jiné úlohy. Ke komunikaci s IMU senzorem LSM6DSO je použita sběrnice I2C a drivery od společnosti STMicroelectronics. , kde byl využit příklad komunikace s SPI a změněn pro komunikaci pro I2C sběrnici. Ve funkci $Init\_LSM6DSO()$ se inicializuje komunikace se senzorem a nastavují měřicí rozsahy akcelerometru a gyroskopu. Akcelerometer má nastavený rozsah na $\pm$ 8 g a gyroskop 2000 dps. Dále jsou níže uvedeny funkce pro čtení a zápis do registrů senzoru. Ve funkcích uvedených níže se ukládají naměřené hodnoty do struktury *Dashboard*. Každá naměřená 16-bitová hodnota je rozdělena do jednotlivých bytů. Struktura obsahuje pole pro každou veličinu. Po uložení bytů do polí se jednotlivé charakteristiky aktualizují pomocí funkce $Custom\_STM\_App\_Update\_Char()$ a jelikož jsou u charakteristik zapnuté notifikace, vyšle mikrokontrolér do mobilní aplikaci informaci o změně charakteristiky. Notifikace je pak zobrazena v

kartě mobilní aplikace. Do mobilní aplikace se zasílají hrubá nezpracovaná data a výpočty probíhají na telefonu. [4]

```
HAL_StatusTypeDef Init_LSM6DSO(void) {
/* Check device ID for LSM6DSO accelerometer */
whoamI = 0;
lsm6dso_device_id_get(&lsm6dsoDriver, &whoamI);
/* Configure LSM6DSO accelerometer */
if (whoamI == LSM6DSO_ID) {
/* Restore default configuration */
lsm6dso_reset_set(&lsm6dsoDriver, PROPERTY_ENABLE);
do {
lsm6dso_reset_get(&lsm6dsoDriver, &rst);
} while (rst);
/* Disable I3C interface */
lsm6dso_i3c_disable_set(&lsm6dsoDriver, LSM6DSO_I3C_DISABLE);
/* Enable Block Data Update */
lsm6dso_block_data_update_set(&lsm6dsoDriver, PROPERTY_ENABLE);
/* Set Output Data Rate */
lsm6dso_xl_data_rate_set(&lsm6dsoDriver, LSM6DSO_XL_ODR_6667Hz);
lsm6dso_gy_data_rate_set(&lsm6dsoDriver, LSM6DSO_GY_ODR_6667Hz);
/* Set full scale */
lsm6dso_xl_full_scale_set(&lsm6dsoDriver, LSM6DSO_8g);
lsm6dso_gy_full_scale_set(&lsm6dsoDriver, LSM6DSO_2000dps);
/* Configure filtering chain */
lsm6dso_xl_hp_path_on_out_set(&lsm6dsoDriver,
    LSM6DSO_LP_ODR_DIV_100);
lsm6dso_xl_filter_lp2_set(&lsm6dsoDriver, PROPERTY_ENABLE);
return HAL_OK;
} else {
return HAL_ERROR;
}

static int32_t lsm6dso_read(void *handle, uint8_t reg, uint8_t *
    bufp, uint16_t len) {

return HAL_I2C_Mem_Read(handle, LSM6DSO_I2C_ADD_H , reg,
    I2C_MEMADD_SIZE_8BIT, bufp, len, 1000);
}
static int32_t lsm6dso_write(void *handle, uint8_t reg, uint8_t
    *bufp, uint16_t len) {
return HAL_I2C_Mem_Write(handle, LSM6DSO_I2C_ADD_H, reg,
    I2C_MEMADD_SIZE_8BIT, bufp, len, 1000);

}
```

Ukázka kódu 4.3: Funkce pro komunikaci s IMU senzorem

```
typedef struct
{
  uint8_t               Vltg_value[4];
  uint8_t             Tmp_value[4];
  uint8_t               IMU_value[6];
  uint8_t       PWM;
  uint8_t              Update_timer_ID;
  uint8_t              Vltg_timer_ID;
  uint8_t              Tmp_timer_ID;
  uint8_t              IMU_timer_ID;
} Dashboard;

Custom_STM_App_Update_Char(CUSTOM_STM_VLTG, (uint8_t *)&data.
    Vltg_value);
Custom_STM_App_Update_Char(CUSTOM_STM_TMP, (uint8_t *)&data.
    Tmp_value);
Custom_STM_App_Update_Char(CUSTOM_STM_IMU, (uint8_t *)&data.
    IMU_value);

void Dashboard_Measurement_Vltg (void)
{
  printf("ADC_Voltage1: %d --- ADC_Voltage2: %d\n",
    ADC_measurement[2],ADC_measurement[3]);
  data.Vltg_value[0] = ADC_measurement[2];
  data.Vltg_value[1] = ADC_measurement[2] >> 8;
  data.Vltg_value[2] = ADC_measurement[3];
  data.Vltg_value[3] = ADC_measurement[3] >> 8;

}

void Dashboard_Measurement_Tmp (void)
{
  printf("ADC_Temperature1: %d --- ADC_Temperature2:     %d\n",
    ADC_measurement[4],ADC_measurement[5]);
  data.Tmp_value[0] = ADC_measurement[4];
  data.Tmp_value[1] = ADC_measurement[4] >> 8;
  data.Tmp_value[2] = ADC_measurement[5];
  data.Tmp_value[3] = ADC_measurement[5] >> 8;
}

void Dashboard_Measurement_Speed (void)
{
    lsm6dso_xl_flag_data_ready_get(&lsm6dsoDriver, &reg_LSM6DSO)
    ;
    if (reg_LSM6DSO) {
    memset(data_raw_acceleration_LSM6DSO.i16bit, 0x00, 3 *
    sizeof(int16_t));
    lsm6dso_acceleration_raw_get(&lsm6dsoDriver,
    data_raw_acceleration_LSM6DSO.i16bit);
    for(int i = 0;i < 6;i++)
      {
        data.IMU_value[i] = lsm6dso_from_fs8_to_mg(
    data_raw_acceleration_LSM6DSO.u8bit[i]);
      }
    }
}
```

Ukázka kódu 4.4: Měření veličin

Kapitola 5

Detekce čar v obraze

Při výpadku připojení je možné použít pro navigaci webkameru pro detekci čar. K detekci byla použita knihovna OpenCV a jazyk Python. Knihovna obsahuje již implementované různé operátory pro předzpracování obrazu a dále transformace, které jsou efektivně implementovány. Předtím než je možné detekovat čáry, je nutné obraz předzpracovat. Nejdříve je na obraz aplikován tzv. Canny edge detektor pro detekci hran, který nejprve pomocí Gaussovského rozmazání potlačí šum, poté se vypočítá gradient obrazu pomocí gradientních masek a nakonec se aplikuje prahování. Pomocí konvoluce obrazu a masky můžeme v obraze například detekovat vertikální nebo horizontální hrany.[15]

−1	−1	−1	−1	0	1
0	0	0	−1	0	1
1	1	1	−1	0	1

Prewitt

−1	−2	−1	−1	0	1
0	0	0	−2	0	2
1	2	1	−1	0	1

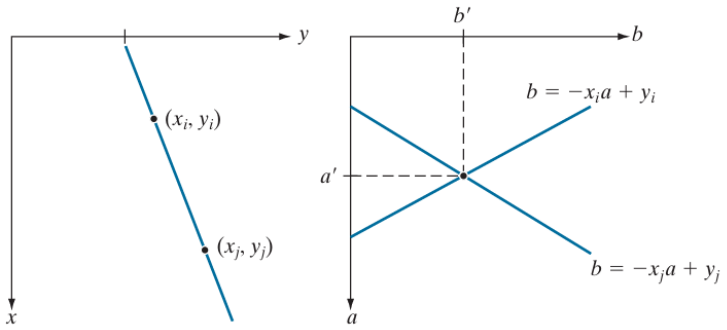
Sobel

Obrázek 5.1: Příklady gradientních operátorů [15]

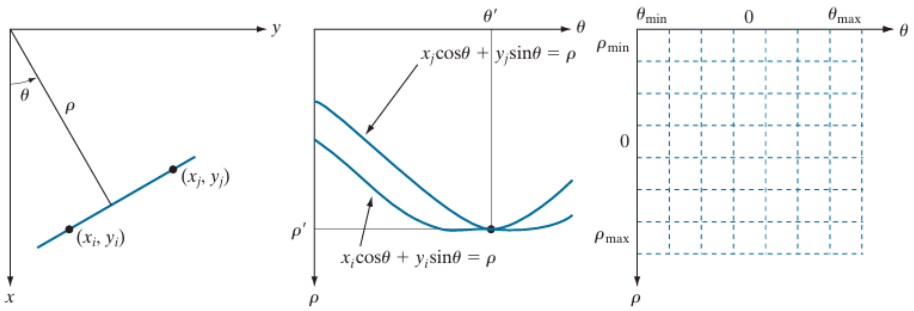
Cesta s čárami se nachází v dolní části obrazu, proto je obraz ořezán na tzv. region of interest (ROI) a další zpracování se provádí pouze na ořezané části. Detekované hrany jsou poté ještě zesíleny. Na takto předzpracovaný obraz už je možné použít Houghovu transformaci, jejíž výstupem v knihovně OpenCV jsou souřadnice čar. Tato transformace převádí bod z prostoru obrazu do prostoru parametrů, kde průnik přímek v prostoru parametrů reprezentuje přímku v prostoru obrazu. Rovnice přímky $y = ax + b$ je normalizována na

rovnici níže, kvůli tomu že vertikální přímky mají nekonečně velkou směrnici a $\rightarrow \infty$. Prostor parametrů θ a ρ je poté diskretizován do tzv. akumulátoru, kde se určí minimální a maximální hodnoty parametrů a jejich krok.[15]

$$xcos(\theta) + ysin(\theta) = \rho \tag{5.1}$$



Obrázek 5.2: Transformace z obrazového prostoru do parametrického [15]



Obrázek 5.3: Normalizace a akumulátor[15]

Výstupních čar je mnoho, a proto je nutné vybrat z nich jednu, která reprezentuje čáru na cestě. Nejprve jsou detekované čáry rozřazené na levou a pravou stranu podle znaménka jejich směrnice. Poté je použita polynomická regrese, kde se počáteční a koncové body čar reprezentují jako množina bodů x a y. Níže jsou uvedené části zdrojového kódu programu, kde jsou vidět jednotlivé kroky zpracování.

```
def display_linesP(image, lines):
    line_image = image

    leftx_array = []
    lefty_array = []

    rightx_array = []
    righty_array = []

    if lines is not None:
        for line in lines:
            for x1,y1, x2,y2 in line:
                slope = (y2 - y1) / (x2 - x1)

                if(slope > 0 and abs(slope) > 0.7):
                    #print('Right' + str(slope))
                    rightx_array.append(x1)
                    rightx_array.append(x2)
                    righty_array.append(y1)
                    righty_array.append(y2)
                    line_image = cv2.line(image, (x1, y1), (x2, y2), (255, 0, 0), 4)
                elif(abs(slope) > 0.7) :
                    #print('Left' + str(slope))
                    leftx_array.append(x1)
                    leftx_array.append(x2)
                    lefty_array.append(y1)
                    lefty_array.append(y2)
                    line_image = cv2.line(image, (x1, y1), (x2, y2), (0, 0, 255), 4)
    #Left
    coeff = np.polyfit(leftx_array,lefty_array, deg = 1)
    a1, b1 = coeff

    y1 = 180
    y2 = 400
    x1 = int((y1 - coeff[1])/coeff[0])
    x2 = int((y2 - coeff[1])/coeff[0])
    line_image = cv2.line(image, (x1, y1), (x2, y2), (85, 246, 38), 10)

    #Right
    coeff = np.polyfit(rightx_array,righty_array, deg = 1)
    a2, b2 = coeff
    y1 = 180
    y2 = 400
    x1 = int((y1 - coeff[1])/coeff[0])
    x2 = int((y2 - coeff[1])/coeff[0])
    line_image = cv2.line(image, (x1, y1), (x2, y2), (85, 246, 38), 10)
```

Ukázka kódu 5.1: Rozřazení čar a polynomická regrese

Velikost obrazu je upravena na rozlišení 640x480 pixelů a barevný prostor je převeden z RGB do stupňů šedé. Dále je aplikován Gaussovský šum s velikostí masky 13x13 pixelů. Metoda *threshold* umožňuje prahování, pokud hodnota pixelu přesáhne hodnoty 150, změní se barva pixelu na bílou (255). Jestliže je hodnota nižší, nastaví se barva pixelu na černou (0). Tato metoda slouží dodatečně k odstranění šumu. Následně je aplikován Canny edge detektor s prahy hysteréze 50 a 200. Pomocí funkce $ROI()$ je vytvořen obrazový frame, který má na dolní části obrazu vykreslený bílý lichoběžník a okolní pixely jsou nulové (černé). Spolu s výstupem $frame\_edge$ z detektoru je aplikována logická funkce AND, která zajistí, že jsou detekované hrany pouze v dolní části obrazu (obrázek 5.5a). Detekované hrany jsou ještě zesíleny funkcí $dilate()$. Posledním bodem je aplikace Houghovy transformace s rozlišením parametrů $\rho = 1$ pixel a $\theta = 1°$ a vykreslení čar v obraze.

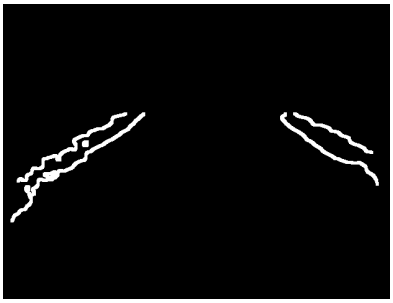
```
1    resize_image =  cv2.resize(frame, (640, 480))
2    colorspace_image = cv2.cvtColor(resize_image, cv2.
     COLOR_RGB2GRAY)
3    frame_blur = cv2.GaussianBlur(colorspace_image,(13,13),
     sigmaX=10)
4
5    ret, thresh1 = cv2.threshold(frame_blur, 150, 255, cv2.
     THRESH_BINARY)
6
7    frame_edge = cv2.Canny(thresh1, 50, 200)
8
9
10   frame_roi = ROI(frame_edge)
11   frame_bitwise = cv2.bitwise_and(frame_edge, frame_roi)
12   img_dilation = cv2.dilate(frame_bitwise, np.ones((3, 3), np.
     uint8), iterations=3)
13   detected_lines = cv2.HoughLinesP(img_dilation, 1, np.pi/180,
     threshold=80,minLineLength=20)
14   frame_final = display_linesP(resize_image,detected_lines)
15
16   cv2.imshow('Line Detection',frame_final)
```

Ukázka kódu 5.2: Výsledné zpracování obrazu

(a) : Vstupní obraz

(b) : Výstup z canny edge detektoru

Obrázek 5.4: Aplikace canny edge detektoru



(a) : Oříznutí a zesílení hran

(b) : Aplikace Houghovy transformace s polynomickou regresí

Obrázek 5.5: Detekce čar

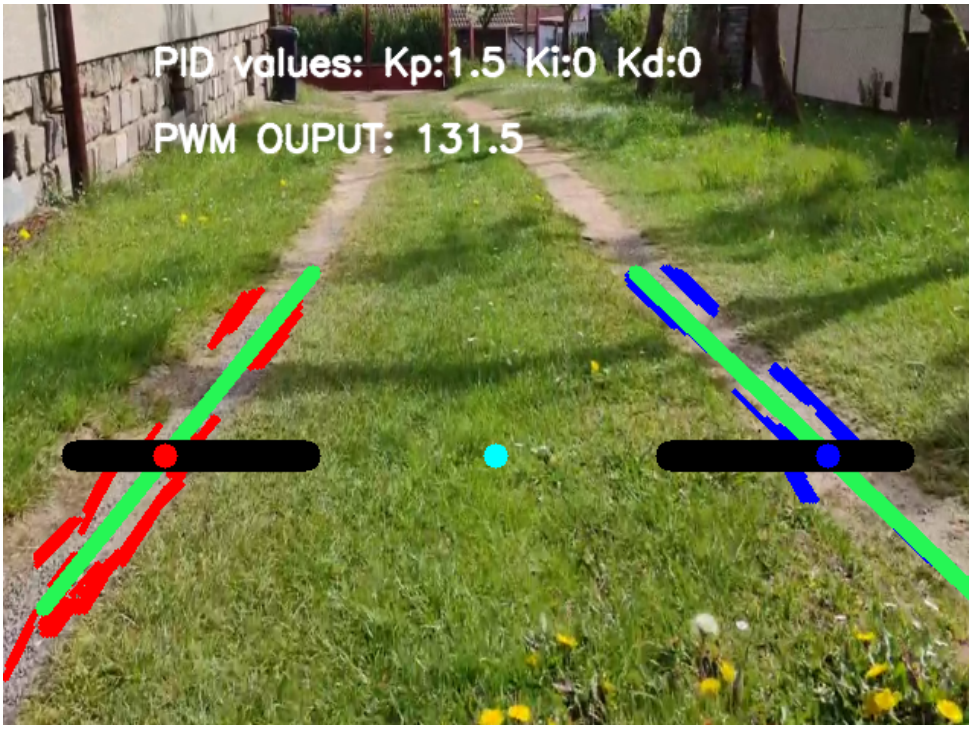
5.1 PID regulátor

Pro sledování čar je použit PID regulátor, který se snaží udržet světle modrý bod uprostřed obrazu. Modrý bod je dán středem mezi červeným a tmavě modrým bodem. Ve funkci $PID()$ se referenční hodnota (pozice středu obrazu) odečítá od aktuální pozice světle modrého bodu. Poté jsou vypočítány složky regulátoru - proporcionální, integrační a derivační. Výstupem funkce je součet jednotlivých složek a offsetu. Offset slouží k tomu, aby servomotor byl v prostřední poloze a kola byla natočená rovně. Proměnná *output* je omezená na šířku pulzů od 1000 μs do 1800 μs. Výstup z regulátoru je zaslán pomocí USB do jednotky, která hodnotu zapíše do CCR registru. Komunikace s řídicí jednotkou probíhá pomocí virtuálního COM portu s knihovnou PySerial.

```
Kp = 1.5
Ki = 1
Kd = 0.3
I = 0
prev_error = 0
setpoint = frame_width/2
offset = 139


```

```
min_steering = 99
max_steering = 179
max_throttle = 200
min_throttle = 179


def PID(Kp, Ki, Kd, I, setpoint, current,prev_error, offset):

    error = setpoint - current
    P = Kp*error
    I = I + Ki*error
    D = Kd*(error - prev_error)

    prev_error = error
    output = offset + P + I + D

    if(output > min_steering):
      output = 179
    elif(output < max_steering):
      output = 99
    print('PWM:' + str(output))
    output_ser = "S" + str(output)
    ser.write(output_ser.encode())
    return output
```

Ukázka kódu 5.3: PID regulátor



Obrázek 5.6: Výstup z programu

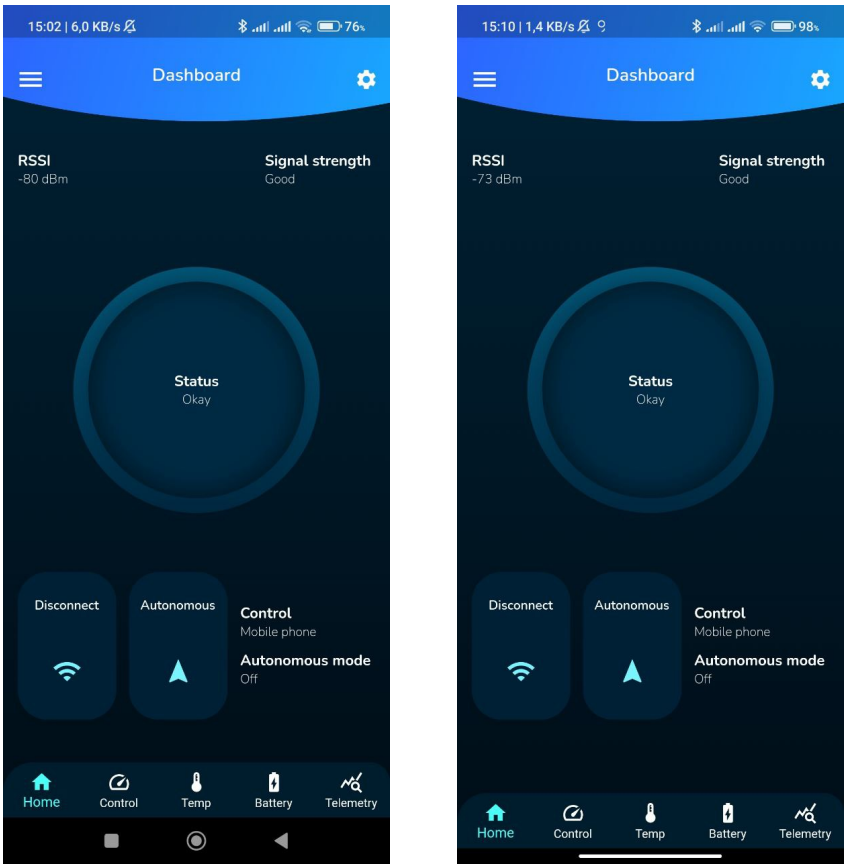
Kapitola 6

Testování RC modelu

6.1 Dosah komunikace

Testování dosahu bylo provedeno na přibližně 32 m (vzdálenost byla zjištěna pomocí Google map). Připojená anténa má zisk 5 dBi. Úroveň RSSI byla při vysílacím výkonu -0.15 dBm na hodnotě -80 dBm, při maximálním vysílacím výkonu 6 dBm se hodnota RSSI pohybovala okolo -73 dBm. Při měření nebyly zaznamenány žádné výpadky a síla signálu byla dostačující.

Obrázek 6.1: Vzdálenost měření



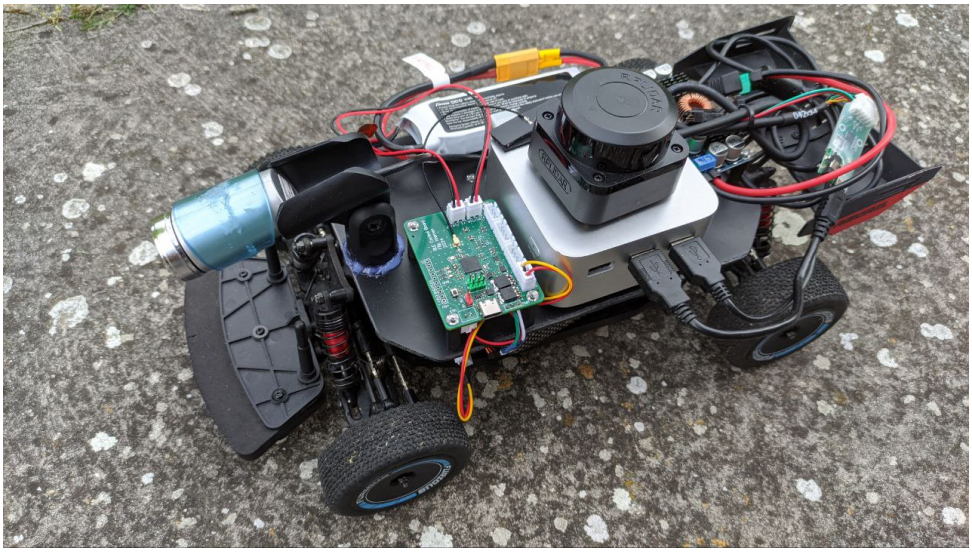
(a) : Vysílací výkon -0.15 dBm **(b) :** Vysílací výkon 6 dBm

Obrázek 6.2: Naměřené hodnoty v aplikaci

6.2 Závěr

Na RC model byla přimontována distančními sloupky plastová deska pro potřebnou elektroniku. Upravený model dále obsahuje DC/DC měnič pro převod napětí 7.4 V na potřebných 12 V pro mini PC. Řídicí jednotka byla navržena v softwaru Altium Designer. Vyrobená deska poté byla osazena a otestována. Byl pro ní napsán obslužný firmware, který měří teplotu akumulátorů a napětí a dále pak komunikuje s akcelerometrem. Pro mobilní telefon byla vytvořena aplikace ve frameworku Flutter, pomocí které jsou zobrazována data ze senzorů a řízen samotný model pomocí dvou virtuálních joysticků. Dále byla implementována detekce čar pomocí OpenCV knihovny, kde se za pomocí PID regulátoru ovládá šířka pulzů PWM modulace servomotoru. Detekci čar je ještě nutné vyzkoušet na reálném RC modelu. K detekci čar ještě chybí přidat skript pro lidar na detekci překážek. Případná vylepšení by spočívala v např. přidání responsivního chování aplikace, tak aby se aplikace přizpůsobila i na jiné velikosti obrazovek. Dále bych chtěl přidat GPS modul pro odhad

rychlosti RC modelu. Nabízí se pro pokročilejší autonomii použít konvoluční neuronovou síť jako klasifikátor. Model by projížděl určitým prostředím a zaznamenával by si trénovací data (snímek cesty a úhel natočení kol). Všechny kódy a návrh desky plošného spoje je možné zhlédnout v příloze práce.



Obrázek 6.3: Zhotovený RC model

Literatura

[1] Rlaarlo [online]. [cit. 2023-12-11]. Dostupné z: https://rlaarlo.com/products/rlaarlo-amx12

[2] AFANEH, Mohammad, 2018. Intro to Bluetooth Low Energy: The easiest way to learn BLE [online]. [cit. 2024-02-25]. ISBN 1790198151. Dostupné z: https://www.novelbits.io/introduction-to-bluetooth-low-energy-book/

[3] PETERKA, RNDr. Ing. Jiří, 2013. Počítačové sítě, v. 3.6: [online]. [cit. 2024-02-25]. Lekce 6: IEEE 802.11 – II. Katedra softwarového inženýrství. Prezentace. Matematicko-fyzikální fakulta, Univerzita Karlova, Praha.

[4] STMICROELECTRONICS. STMems_Standard_C_drivers [online]. [cit. 2024-05-11]. Dostupné z: https://github.com/STMicroelectronics/STMems_Standard_C_drivers

[5] STMICROELECTRONICS. Sequencer [online]. [cit. 2024-03-02]. Dostupné z: https://community.st.com/ysqtg83639/attachments/ysqtg83639/mcu-wireless-forum/6301/1/STM32WB_BLE_SW_Sequencer.pptx

[6] ESC Protocols [online]. [cit. 2024-04-19]. Dostupné z: https://docs.px4.io/main/en/peripherals/esc_motors.html

[7] STMICROELECTRONICS. NUCLEO-WB55RG [online]. [cit. 2024-04-19]. Dostupné z: https://www.st.com/en/evaluation-tools/nucleo-wb55rg.html

[8] STMICROELECTRONICS, 2023. How to develop RF hardware using STM32WB microcontrollers [online]. [cit. 2024-05-12]. Dostupné z: https://www.st.com/resource/en/application_note/an5165-how-to-develop-rf-hardware-using-stm32wb-microcontrollers-stmicroelectronics.pdf

[9] MIOLA, Alberto, 2020. Flutter Complete Reference: Create beautiful, fast and native apps for any device [online]. [cit. 2024-03-11]. ISBN 979-8691939952. Dostupné z: doi:https://www.amazon.com/Flutter-Complete-Reference-Create-beautiful/dp/B08KH3R9SY

[10] YOGITA KUMAR, 2020. Flutter: Stateful Widgets vs Stateless Widgets [online]. [cit. 2024-05-19]. Dostupné z: https://levelup.gitconnected.com/flutter-stateful-widget-vs-stateless-widget-c7baf0a3bbc1

[11] VISHAY, 2015. How to select an NTC Thermistor [online]. [cit. 2024-05-12]. Dostupné z: https://www.vishay.com/docs/33001/seltherm.pdf

[12] VISHAY. NTC RT CALCULATION TOOL [online]. [cit. 2024-05-02]. Dostupné z: https://www.vishay.com/en/thermistors/ntc-rt-calculator/

[13] CURVE FIT. Curve Fit [online]. [cit. 2024-05-12]. Dostupné z: https://curve.fit/

[14] Lipo Voltage Chart [online], 2018. [cit. 2024-05-12]. Dostupné z: https://blog.ampow.com/lipo-voltage-chart/

[15] GONZALEZ, Rafael C. a Richard E. WOODS, 2018. Digital Image Processing [online]. 4. Pearson [cit. 2024-05-11]. ISBN 978-1-292-22304-9. Dostupné z: https://www.pearson.com/en-us/subject-catalog/p/digital-image-processing/P200000003224/9780137848560

Příloha A

Zdrojové kódy pro mobilní aplikaci, firmware pro řídicí modul a Python aplikaci pro detekci čar

Dostupné z:
https://gitlab.fel.cvut.cz/kremefil/tweaky-flutter
https://gitlab.fel.cvut.cz/kremefil/tweaky-stm32
https://gitlab.fel.cvut.cz/kremefil/line-detection-in-opencv

Příloha B

Návrh DPS

Dostupné z: https://gitlab.fel.cvut.cz/kremefil/ble-board-pcb