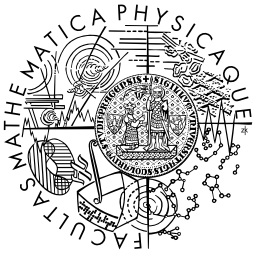


MATEMATICKO-FYZIKÁLNÍ
FAKULTA
Univerzita Karlova

BAKALÁŘSKÁ PRÁCE

Daniel Kurek

Framework pro interaktivní distribuovanou síť světelných zařízení

Katedra teoretické informatiky a matematické logiky

Vedoucí bakalářské práce: RNDr. David Obdržálek, Ph.D.

Studijní program: Informatika

Praha 2024

Prohlašuji, že jsem tuto bakalářskou práci vypracoval(a) samostatně a výhradně s použitím citovaných pramenů, literatury a dalších odborných zdrojů. Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona v platném znění, zejména skutečnost, že Univerzita Karlova má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle §60 odst. 1 autorského zákona.

V Praze dne 09.05.2024 Daniel Kurek

V první řadě bych chtěl poděkovat vedoucímu práce panu RNDr. Davidu Obdržálkovi, Ph.D. za cenné rady a trpělivost během vypracování této práce. Dále bych chtěl poděkovat celé rodině za podporu v celém průběhu studia.

Název práce: Framework pro interaktivní distribuovanou síť světelných zařízení

Autor: Daniel Kurek

Katedra: Katedra teoretické informatiky a matematické logiky

Vedoucí bakalářské práce: RNDr. David Obdržálek, Ph.D., Katedra teoretické informatiky a matematické logiky

Abstrakt: Tato práce představuje framework pro práci s distribuovanou sítí světelných zařízení založených na modulech Espressif ESP32–S3. Framework lokalizuje zařízení na základě měření jejich vzájemných vzdáleností, přičemž vzhledem k plánovanému použití frameworku rozlišuje mezi pevně umístěnými stanicemi a mobilními zařízenimi. Součástí práce je i pomocná aplikace pro osobní počítač, která vizualizuje lokalizaci a umožňuje přímé ovládání (nastavení) zařízení. Pro komunikaci framework používá Bluetooth mesh a pro měření vzdáleností WiFi FTM (RTT). K otestování frameworku byla navržena a vyrobena fyzická zařízení. Framework a vyrobená zařízení jsou předvedeny na hře 'Najdi svoji barvu', kde je cílem dojít s mobilním zařízením k pevně umístěné stanici se stejnou barvou, jako má mobilní zařízení.

Klíčová slova: Bluetooth mesh, lokalizační systém, distribuovaná síť

Title: Framework for interactive distributed network of luminous devices

Author: Daniel Kurek

Department: Department of Theoretical Computer Science and Mathematical Logic

Supervisor: RNDr. David Obdržálek, Ph.D., Department of Theoretical Computer Science and Mathematical Logic

Abstract: This work presents a framework for interactive appliaction on a distributed network of lighting devices based on Espressif ESP32–S3 modules. The framework locates devices by measuring their mutual distances. There are two types of devices, fixed stations and mobile devices, due to the intended use of the framework. The work also includes a helping application for a PC that visualizes the localization and allows direct control of devices. The framework uses Bluetooth mesh for communication and WiFi FTM (RTT) for distance measurement. To test the framework, physical devices were designed and fabricated. The framework and fabricated devices are demonstrated in a 'Find Your Color' game where the goal is to bring mobile device to a fixed station with the same color as the mobile device.

Keywords: Bluetooth Mesh, Localization System, Distributed Network

Obsah

Úvod

Cílem této práce je navrhnout a implementovat framework pro práci se sítí světelných zařízeních, které jsou rozmístěné po ploše 50×50 m. Začneme cílovou aplikací a postupně si rozebereme, z čeho se bude framework skládat.

Cílová aplikace Na ploše jsou rozmístěná světelná zařízení. Každé zařízení svítí nějakou barvou. Uživatelé se mezi nimi budou pohybovat se zařízením v ruce, na kterém bude také svítit barevná dioda. Každý uživatel bude mít za úkol najít na dané ploše stejnou barvu, jakou má na zařízení, které si nese, a přijít k němu. Jakmile hra zaznamená, že stojí u správné barvy, zobrazí se uživateli nová barva, kterou má uživatel hledat.

Z toho nám vyplývá několik důležitých věcí, co zařízení musí dokázat poskytovat. Zařízení budou mezi sebou komunikovat, aby si mohla zjistit, jaké jsou barvy ostatních zařízení a abychom mohli jednoduše nastavit počáteční barvy. Zařízení také musí umět detekovat, jak blízko je k ostatním zařízením. Může to být buď binární hodnota 'jsem blízko', nebo to může být nějaké číslo, které nám určí vzdálenost.

Navíc v rámci frameworku bude lokalizační systém, který dokáže určit polohy zařízení v rámci sítě. Tento lokalizační systém by měl být soběstačný, bez potřeby externí infrastruktury. Takový systém by se mohl využít na zobrazení mapy uživateli, která by ukazovala, kde se právě nachází a kam má dojít, nebo na chytré rozmístění barev.

Zařízení budou rozdělena na mobilní a stacionární (neboli stanice). Každý uživatel bude mít jedno mobilní zařízení, stanice budou napevno umístěné v prostoru.

Pro použití frameworku bude potřeba mít více zařízení. Výběr a návrh zařízení zohlední i cenovou dostupnost výsledné aplikace.

Framework bude testován na 10 zařízeních, ale navrhován bude i na větší počet zařízení. Ideálně by měl zvládat 30 uživatelů a 20 stanic.

Práce je rozdělena do 5 kapitol. V první kapitole se budeme věnovat výběru zařízení a jednotlivých technologií, které budou zařízení využívat. Ve druhé kapitole navrhneme strukturu frameworku a popíšeme technickou implementaci. Třetí kapitola se bude zabývat konstrukcí zařízení. Ve čtvrté kapitole si ukážeme příkladové programy využívající náš framework. V páté kapitole se podíváme na výsledky testu reálné implementace na malém počtu zařízení.

1 Analýza problému a výběr metod řešení

V této kapitole se nejprve budeme zabývat výběrem zařízení. Dále se podíváme na technologie a možná řešení, které můžeme použít pro komunikaci a měření vzdálenosti mezi zařízeními a následně i pro vlastní lokalizaci.

1.1 Výběr zařízení

Existují dva hlavní druhy zařízení, která můžeme použít, buď mikrokontroléry nebo klasické počítače. Cílová cena za jedno zařízení by měla být co nejnižší, nejlépe do 1000 Kč, aby se mohlo jednoduše uskutečnit i větší nasazení frameworku. Z tohoto důvodu vyřadíme většinu klasických počítačů, jelikož jejich cena se pohybuje v řádu tisíců Kč. Dále zařízení musí podporovat bezdrátové sítě a připojení periferií, např. LED, tlačítek, atd.

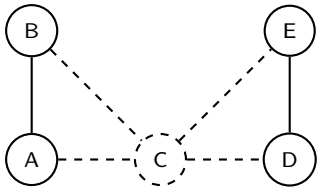
Mikrokontrolér, též jednočipový počítač, je speciální typ čipu, který v sobě zahrnuje vše potřebné k provozování aplikace. Typicky obsahuje procesor, operační paměť, paměť pro uložení programu a další podpůrné obvody (napájení, hardwarové periferie atd.). Občas je paměť pro uložení programu zvlášť, aby se mohly jednoduše vyrábět varianty mikrokontrolérů s různými velikostmi pamětí. Hlavní výhodou mikrokontrolérů je jejich jednoduchost a cena. Cena se většinou pohybuje okolo pár desítek korun v jednotkovém množství. Okolo mikrokotrolérů se staví také tzv. System on a chip (SoC), který integruje další moduly do jednoho čipu, např. WiFi, Bluetooth, …

Výše zmíněným kritériím vyhovuje mnoho zařízení. Budeme ovšem vybírat pouze z pár kandidátů, jenž mají potřebnou funkčnost a se kterými jsem obeznámen, konkrétně Raspberry Pi Zero, rodina ESP32 čipů a rodina nrf52xxx a nrf53xx čipů. Raspberry Pi Zero není dostupné ve větším množství během psaní této práce, takže je nebudeme zvažovat. Čipy nrf52xxx a nrf53xx vyhovují veškerým parametrům, nicméně rodina ESP32 je výkonnější a disponuje větší pamětí a navíc jsou levnější, tedy pro tuto práci zvolime rodinu čipů ESP32.

V rodině ESP32 je celá řada zařízení, přičemž každé je přizpůsobené jinému účelu. Nejsou mezi nimi moc velké cenové rozdíly, takže si vybereme ty nejvýkonnější — ESP32-S3. Později můžeme přejít i na jiné zařízení od firmy Espressif, jelikož jejich framework na vývoj aplikací (ESP-IDF [1]) je kompatibilní mezi jejich zařízeními. Samozřejmě pokud dané zařízení disponuje stejnými prostředky, které v kódu využíváme. Např. kód na Bluetooth nám nebude fungovat na zařízení, které nepodporuje Bluetooth komunikaci. ESP32-S3 disponuje bohatými periferiemi, 45 GPIO (General-purpose input/output) pinů z toho 28 pinů můžeme použít bez větších omezení na komunikaci s jinými čipy nebo na ovládání LED nebo tlačítek.

1.2 Komunikace

Propojení zařízení se může vyřešit dvěma způsoby, buď pomocí centralizované sítě nebo pomocí distribuované sítě. V této sekci si tyto sítě porovnáme a vybereme



Obrázek 1.1 Příklad mesh sítě kde po výpadku jednoho zařízení `C` se síť rozdělí na dvě nezávislé sítě, `AB` a `DE`.

jaký typ sítě je lepší pro naší aplikaci. Následně vybereme konkrétní implementaci sítě.

Centralizovaná síť Centralizované propojení, např. WiFi, vyžaduje, aby jeden access point (AP) dosáhl na všechny zařízení. To se může vyřešit propojením více AP mezi sebou a tím pokrýt všechny zařízení. To pro naší aplikaci není příliš vhodné, protože bychom museli pro každé prostředí testovat, jestli je pokrytí dostatečné a případně přesouvat AP. Navíc pokud by nám vypadl jeden AP, tak by to výrazně ovlivnilo funkčnost aplikace. Komunikace by k části zařízení úplně vypadla.

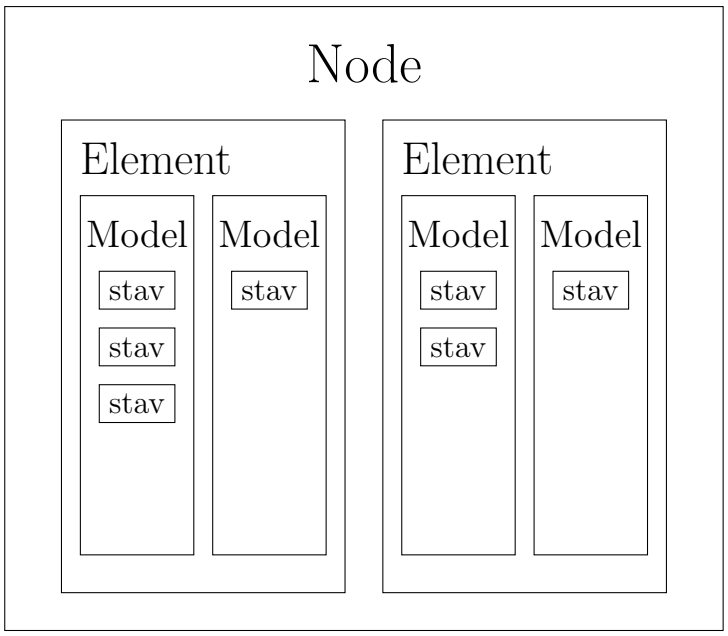
Distribuovaná síť Distribuované propojení, konkrétně *mesh síť*, nám řeší problémy centralizované sítě. Jednotlivá zařízení se propojí mezi sebou a tím utvoří síť. Takže pokrytí sítě nemusíme moc řešit, protože nám stačí zajistit, aby každé zařízení dokázalo komunikovat s alespoň jedním dalším. Výpadek jednoho zařízení nám chod všech zařízení neovlivní, v nejhorším případě se síť rozpadne na více částí (viz. obrázek 1.1).

S distribuovaném propojení nám ale vzniká i několik jiných problémů. Např. mesh sítě mají složitější routing. Zařízení se můžou odpojit nebo připojit kdykoliv. Takže pokud nám probíhá komunikace mezi dvěma zařízeními a nějaké zařízení na cestě se odpojí, tak musí síť zajistit opravu této cesty, aby komunikace mohla pokračovat. Další problém je s datovou propustností sítě. Typicky je mnohem nižší než u centralizované sítě [2], protože udržování spojení v mesh síti je typicky složitější a navíc většinou zprávy nejdou optimální cestou.

Pro náš framework se jeví jako nejvhodnější mesh síť, protože typická aplikace bude v prostředí, kde nemáme připravenou žádnou infrastrukturu. Navíc je jednoduší zaručit dostatečné pokrytí sítě.

Mesh sítí je několik, které můžeme použít na našich zařízeních. Většina těchto sítí je určená pro IoT zařízení nebo zařízení pro chytrou domácnost. Zejména můžeme vybírat mezi WiFi Mesh [3], Bluetooth mesh [4], které jsou součástí ESP-IDF.

WiFi Mesh ESP-WiFi-Mesh je mesh síť postavená na obyčejném fungování WiFi sítě. Na aplikační úrovni se přidá logika, která zajišťuje propojení samotných zařízení a routing mezi nimi. Každé zařízení vytvoří access point a zároveň se připojí k jinému zařízení (kromě jednoho, které pouze vytvoří AP). WiFi Mesh takto vytvoří stromovou strukturu zařízení, kde každý vrchol představuje zařízení, které vytvořilo access point. Hrany pak představují připojení zařízení k access



Obrázek 1.2 Struktura Bluetooth mesh jednoho zařízení. Překresleno z [8, s. 6]

pointu jiného zařízení. WiFi Mesh řeší i self-healing sítě, které zabezpečuje správné fungování sítě i při výpadku zařízení ze sítě.

V dokumentaci implementace sítě na naše zařízení je změřený výkon sítě [3, Performace]. Největší problém pro náš účel je vysoký čas na obnovu výpadku zařízení (healing time). Čas obnovy sítě pro kořen stromu je $< 10\,\mathrm{s}$ a pro ostatní zařízení to je $< 5\,\mathrm{s}$. Pokud by se mobilní zařízení pohybovalo rychle, tak by mohlo být většinu času odpojené od sítě a nemohli bychom s ním komunikovat. To se nám nehodí do sítě, kde se některé zařízení mohou neustále pohybovat. Z tohoto důvodu není tato síť vhodná pro naše účely.

Bluetooth mesh *Bluetooth mesh* je mesh síť od Bluetooth SIG, která vyvíjí Bluetooth protokol. Síť je postavená na Bluetooth Low Energy protokolu [5]. Na rozdíl od WiFi Mesh je Bluetooth mesh síť typu flooding. Odeslanou zprávu přijmou všechna okolní zařízení, která zprávu přepošlou dál. Jednotlivá zařízení si tedy neuchovávají informace o nejlepší cestě k cílovému zařízení. Tento mechanismus je vhodný pro naše použití, protože síti nevadí, když se zařízení rychle pohybují nebo když nějaké zařízení vypadne [5, otázka: Is traffic affected if nodes break?].

Ostatní sítě Dále existují i jiné distribuované sítě, např. Thread [6] a Zigbee [7]. Podle srovnávací tabulky [2, s. 3] obsahují router zařízení, tedy může nastat podobný problém jako u WiFi Mesh s dlouhou obnovou sítě. Pro naše účely tedy zvolíme siť Bluetooth mesh, kterou v následujícím textu detailněji představíme.

Úvod do Bluetooth mesh

V této sekci si vysvětlíme základy Bluetooth mesh. Pro další podrobnosti se můžeme odkázat na *Bluetooth Mesh Networking: The Ultimate Guide* [9] nebo na *The Fundamental Concepts of Bluetooth Mesh Networking* [10][11].

Bluetooth mesh je konkrétně tzv. 'managed-flooding' síť. Flooding síť pošle každý paket všem okolním zařízením a každé zařízení přepošle paket dál. Tím si můžeme být jistí, že pokud je v síti zařízení, kterému byl paket adresován, tak jej dostalo.

Pokud složení sítě je kombinace zařízení, které jsou napájeny z baterie a které jsou napájeny ze sítě, tak můžeme flooding síť vylepšit. Výměna baterií v zařízení nebo jeho nabití může být časově náročné, hlavně pokud bychom to měli dělat každý den. K tomu slouží ta managed část v managed-flooding. Můžeme totiž dát schopnost přeposílání packetů pouze některým zařízením. V Bluetooth mesh jsou také i jiné způsoby šetření energie zařízením napájeným z baterií. Zařízení může být tzv. Low Power Node, což je zařízení, které se může vypnout na nějakou dobu. Všechny packety, které jsou určeny pro takové zařízení, bude udržovat zařízení typu Friend, které je typicky napájené ze sítě. Tuto funkci využívat nebudeme, protože všechna zařízení budou napájena z baterie.

Hlavní výhodou Bluetooth mesh je kompatibilita produktů mezi různými výrobci. Veškerá komunikace v síti je v rámci pevně definovaných *modelů*. Model definuje strukturu zprávy a příkazy, které můžeme provést. Bluetooth SIG definovalo několik modelů, které je doporučeno používat [8]. Jednotlivý výrobci si můžou definovat i vlastní modely. Vlastní modely by se měli používat pouze v případě, když neexistuje model od Bluetooth SIG, který můžeme využít.

Modely Modely jsou rozděleny na server a klient. Server model uchovává data a provádí akce na základě těchto dat. Zatímco klient model mění uložená data, která jsou na serveru. Například GenericOnOff model definuje jediný stav, a to jednoduchou binární hodnotu. U tohoto modelu si server můžeme představit jako žárovku, která je buď zapnutá nebo vypnutá, a klient model si představíme jako vypínač, který ovládá stav žárovky (buď pošle příkaz ON nebo OFF).

Elementy Modely nemůžeme přidat přímo na zařízení. Musíme je umístit do *elementů*. Element je kolekce modelů, která dostane jednu unikátní adresu. V jednom elementu nemůžeme mít více stejných modelů, protože potom bychom je nemohli navzájem mezi sebou rozlišit. Pokud chceme více modelů jednoho typu na zařízení, tak je dáme do různých elementů, poté jednotlivé modely rozlišíme pomocí adres elementů. Počet elementů na jednom zařízení není omezen, jediné omezení je počet adres k přidělení.

Adresy Bluetooth mesh adresy jsou čísla o 16 bitech, která jsou typu unsigned integer. Některé adresy mají speciální význam, např. `0xFFFF` znamená, že se správa pošle všem zařízením. Zařízení se skládá z elementů (viz. obrázek 1.2), kde každý element má přiřazenu jednu adresu. Jedno zařízení tedy může mít i více než jednu adresu. Adresy jsou přiřazovány v rámci zařízení sekvenčně, tedy zařízení má přiřazenou tzv. `base address` a každý element má poté adresu `base address` $+ i$, kde i je index elementu (první index je 0).

Provisioning Další důležitá část Bluetooth mesh je zprovoznění sítě. Na rozdíl od centralizovaných sítí, na které jsme běžně zvyklí, nemůžeme pouze na jednom místě vytvořit síť a jednotlivým zařízením předat pouze přihlašovací údaje. V Bluetooth mesh potřebujeme zařízení, které zabezpečuje nastavení jednotlivých

zařízení, toto zařízení se nazývá provisioner. Může to být jakékoliv Bluetooth zařízení s vhodným programem, i smartphone se speciální aplikací. Provisioner zařízení přidává jednotlivá zařízení do sítě. V tomto procesu přiřadí zařízením jejich adresy a klíče pro komunikaci v síti. Pokud využíváme více provisioner zařízení, tak se musíme postarat o to, že klíče budou stejné a přiřazované adresy budou unikátní. Když poprvé spustíme zařízení, tak je ve stavu unprovisioned a tuto informaci vysílá ostatním. Jakékoliv provisioner zařízení jej uvidí a může zahájit proces přidávání do sítě (provisioning).

Klíče Pro správné fungování zařízení v síti musí mít dva typy klíčů, pomocí kterých bude poté komunikovat s ostatními, Network key a App key. Tyto klíče jsou předány zařízení při provisioning procesu. Network key slouží na šifrování komunikace v síti. Tento klíč musí mít všechna zařízení stejný. App key slouží k autentifikaci jednotlivých aplikací. Váže se na jednotlivé modely na zařízení. Pokud chce klient model komunikovat se server modelem, tak musí mít oba klíče stejné. Bez nich nemůžeme s modelem komunikovat. App key v praxi slouží na oddělení zařízení, např. aby zařízení na ovládání světel nemělo přístup k otevíráním dveří.

Publish Další věcí v Bluetooth mesh je publish funkce. Pomocí publish zpráv může server model poslat zprávu o svém stavu, aniž by si to jiné zařízení vyžádalo. Publish zprávy využívají modely také na odpovídání na příkazy získaní stavu při použití adresy označující více zařízení. Navíc můžeme nastavit, aby server pravidelně posílal zprávy o jeho stavu, popřípadě aby je poslal pouze v případě změny stavu.

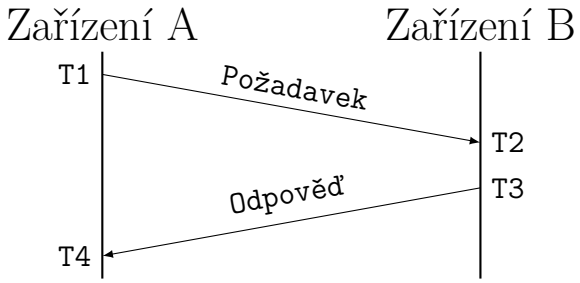
1.3 Měření vzdálenosti

Existuje několik různých způsobů měření vzdálenosti založených na bezdrátových technologiích [12]. Vybraná zařízení podporují dva nejčastější způsoby, a to *síla signálu* a *Time-of-Flight*.

Síla signálu Naše zařízení dokáže měřit indikaci síly rádiového signálu (RSSI), který přijímá. Rádiový signál, podobně jako světlo, slábne s uraženou vzdáleností. Míra ztráty na intenzitě záleží na médiu, kterým se šíří. Tedy když známe sílu původního signálu a vlastnosti média, tak si můžeme vypočítat jakou vzdálenost urazil. RSSI si můžeme vyjádřit následující rovnicí.

$$\mathrm{RSSI} = -10n \lg \frac{d}{d_0} + A \tag{1.1}$$

kde d je vzdálenost od zařízení, d_0 je vzdálenost naměřené hodnoty RSSI A a n je parametr ztráty signálu daného prostředí [13]. Ve statickém prostředí tato metoda může dosahovat přijatelných výsledků. Jakmile se prostředí mění, tak se na tuto metodu nemůžeme spoléhat. Rozdíl ztráty signálu mezi vzduchem a pevnými objekty je příliš velký. Tudíž měření vzdálenosti by zásadně ovlivnilo i to, jestli člověk projde mezi zařízeními, které právě mezi sebou měří vzdálenost.



Obrázek 1.3 Měření vzdálenosti pomocí Time-Of-Flight metody. Zařízení A vyšle požadavek zařízení B a zaznamená si čas odeslání (`T1`). Zařízení B odpoví na zprávu s časem přijmu požadavku (`T2`) a časem odeslání odpovědi (`T3`). Zařízení A si zaznamená čas příjmu odpovědi (`T4`). Pokud by se od zařízení B signál odrazil sám, poté by `T2` a `T3` bylo stejné. Výsledná vzdálenost se spočítá jako $d = (v \times (\texttt{T4} - \texttt{T1}) - (\texttt{T3} - \texttt{T2}))/2$, kde v je rychlost šíření signálu.

Time-of-Flight Time-of-Flight metoda, jak lze odvodit z názvu, měří čas strávený signálem během jeho cesty mezi dvěma body. To můžeme změřit tak, že si zaznamenáme čas, kdy signál vyrazil z bodu A a kdy dorazil do bodu B. Abychom měli přesné měření, tak musíme mít synchronizované hodiny v bodě A s bodem B. Při malých rychlostech signálu je možné zajistit dostatečně přesnou synchronizaci, aby měření vzdálenosti neovlivnila, při rychlostech blížící se rychlosti světla toto už zaručit nemůžeme. Proto se většinou měří za jakou dobu signál dorazí z bodu A do bodu B a zpátky, v některých případech odraz signálu zpět není instantní. Tímto způsobem nemusíme mít synchronizované hodiny, ale stačí nám pouze hodiny, které měří přesně čas.

Senzory měření vzdálenosti od pevných objektů využívající Time-of-Flight spoléhají na to, že se signál odrazí od objektu díky fyzikálním jevům, což je instantní. My potom změříme za jakou dobu se signál vrátil zpět. V našem případě zařízení musí signál přijmout, zpracovat a až poté odeslat odpověď zpátky (viz. obrázek 1.3). Jelikož se na odrazu signálů podílí procesor, který vykonává i jiný kód, tak odraz signálů není konstantní. Tím se vnese chyba do měření.

Tento způsob měření vzdálenosti závisí na znalosti přesné rychlosti šíření signálu, v našem případě rádiového signálu, a přesné měření času. Rádiový signál se ve vakuu šíří rychlostí světla, ale v jiných prostředích se šíří pomaleji. Ve vzduchu je rychlost téměř stejná jako ve vakuu. Problém nastává v přesném měření času. Když naše zařízení měří čas pomocí procesoru, tak rozlišení časovače je $1/(240 \cdot 10^6)$ s. Můžeme si tedy vypočítat jakou chybu zavedeme do měření, pokud bychom se zpozdili o jeden takt našeho procesoru v celém procesu měření.

$$d = \frac{c}{f_{CPU}} = \frac{299792458}{240000000} \approx 1{,}24\,\text{m} \tag{1.2}$$

Za jeden takt našeho procesoru urazí signál zhruba 1,24 m (dle 1.2).

V ESP-IDF můžeme vzdálenost měřit pomocí *WiFi FTM* (FTM=Fine Timing Measurement), také WiFi-RTT (Round-Trip-Time), což využívá Time-of-Flight metodu. Tento proces je velmi náchylný k neočekávaným zpožděním, takže WiFi FTM tento proces několikrát v rámci jednoho měření opakuje a zprůměruje jednotlivá měření. Měření vzdálenosti je rozděleno na dávky jednotlivých měření. Mezi dávky je časový odstup podle konfigurace. Můžeme si nastavit o kolik

jednotlivých měření se zařízení pokusí a časový odstup mezi dávky. Jedna dávka je vždy 4 nebo 8 měření, takže celkový počet měření musí být celočíselný násobek těchto dávek.

V práci budeme tedy využívat Time-of-Flight metodu pomocí WiFi FTM, jelikož měření jsou přesnější než pomocí metody síly signálu.

1.4 Lokalizační systémy

Lokalizační systém se používá k určování polohy zařízení. Běžně používané globální systémy využívají satelity pro určení polohy. Pro správné fungování satelitních systému musí signál ze satelitů dorazit až k našemu zařízení. Pokud není zaručená přímá viditelnost na satelity, tak přesnost určení polohy je nižší nebo v nejhorších případech polohu nejde určit vůbec. Pro naše účely je vhodnější lokalizační systémy využívající pouze naše zařízení na určení polohy. Takový systém nám nabízí větší volnost v prostorech, kde aplikaci můžeme používat.

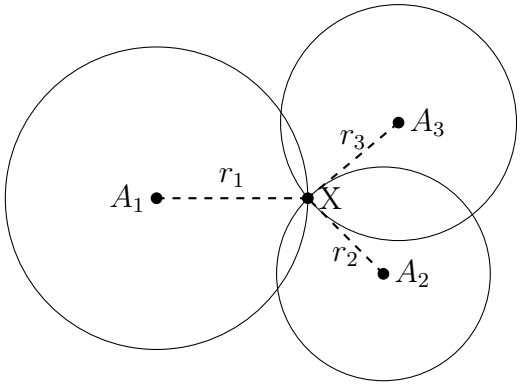
Hlavní dvě metody na lokalizaci, které na zařízeních mohou být využita, jsou *fingerprinting* a *laterace*. Metody v následujícím textu budou představeny a následně bude vybrána metoda pro naší aplikaci.

Fingerprinting Fingerprinting metoda se skládá ze dvou fází, sbíraní dat a určovaní polohy. Pro každou polohu, kterou chceme umět určit, zaznamenáme data, která nám jednoznačně určují danou polohu. Většinou se zaznamenává síla signálu WiFi AP, nebo jiných vysílačů, případně se může využít i WiFi FTM. Poté tyto data zpracujeme a nahrajeme na jednotlivá koncová zařízení. Tato zařízení poté budou měřit stejná data a podle nich určí k jaké poloze z prvního kroku jsou nejblíže a tuto pozici prohlásí jako jejich aktuální. K určení nejbližší polohy můžeme využít různé algoritmy ze strojového učení, např. kNN (k nejbližších sousedů) nebo SVM (Support vector machine) [14]. Výhody této metody jsou, že tato metoda je jednoduchá a může být použita na mnoha různých zařízeních. Hlavní nevýhodou je, že pro každé prostředí, i pro každou změnu konfigurace prostředí, musíme znovu provést náročný první krok získávání dat.

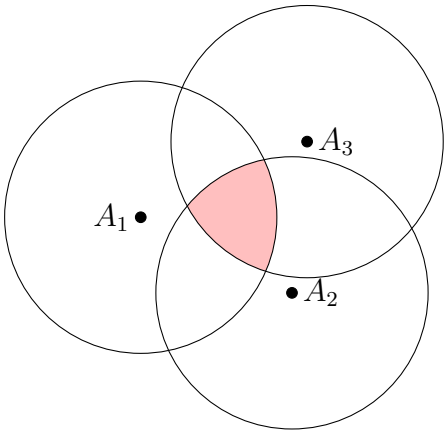
Laterace Laterační metoda určuje polohy ze vzdáleností od bodů, u kterých známe přesnou polohu, tyto body budeme nazývat kotvami. Tato metoda se hlavně používá u GPS. Laterace je založená na geometrickém významu měření vzdáleností. Pro 2D prostor si měření vzdálenosti reprezentujeme jako kružnice se středy pozicích kotev a poloměry podle naměřené vzdálenosti[1]. Poté je poloha, ze které se vzdálenosti měřili, v průsečíku kružnic. Na jednoznačné určení polohy potřebujeme alespoň 3 kružnice[2]. V reálném světě budeme mít nepřesnosti v jednotlivých měření vzdáleností. Může se nám tedy stát, že se kružnice neprotnou v jednom bodě, poté máme pouze nějakou oblast, ve které se nacházíme. V takovém případě můžeme najít bod v dané oblasti, který má nejmenší chybu vůči všem měřením. Na to se může využít metoda nejmenších čtverců [15].

[1]Ve 3D prostoru bychom měření reprezentovali jako koule.

[2]Může nastat speciální případ, kdy se dvě kružnice budou dotýkat pouze v jednom bodě. Poté bychom už mohli vědět přesnou polohu.



Obrázek 1.4 Příklad multilaterace, kde A_1 – A_3 jsou pevně umístěné body, u kterých známe jejich pozice. r_1 – r_3 jsou naměřené vzdálenosti.



Obrázek 1.5 Příklad multilaterace, kde vzdálenosti od pevných bodů A_1 – A_3 jsou nepřesně naměřeny. Výsledná poloha může být kdekoliv uvnitř vybarvené oblasti.

Pro cílové využití je vhodnější laterační metoda, jelikož u fingerprinting metody bychom museli při každé změně rozložení stanic znovu natrénovat lokalizační systém.

Multilaterace

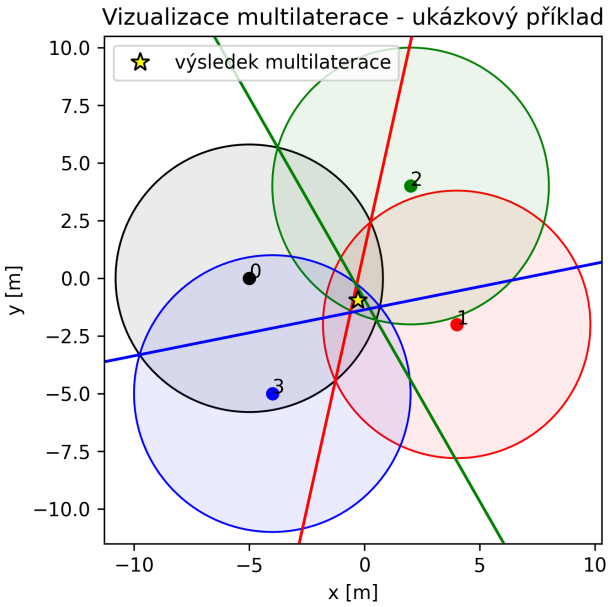
Pro určení polohy zařízení pomocí multilaterace potřebujeme alespoň 3 body, u kterých známe jejich polohu a vzdálenost k nim 1.4. Určení polohy můžeme zapsat jako soustavu rovnic.

$$(x_1 - x_0)^2 + (y_1 - y_0)^2 = r_1^2 \tag{1.3}$$
$$(x_2 - x_0)^2 + (y_2 - y_0)^2 = r_2^2 \tag{1.4}$$
$$(x_3 - x_0)^2 + (y_3 - y_0)^2 = r_3^2 \tag{1.5}$$

V rovnicích známe vše kromě x_0 a y_0. Systém rovnic si převedeme do standardního zápisu lineárních rovnic $Ax = b$. Odečtením první rovnice 1.3 od druhé 1.4 dostaneme po úpravách následující tvar:

$$x_0(x_2 - x_1) + y_0(y_2 - y_1) = \frac{1}{2}\left(r_1^2 - r_2^2 + (x_2^2 + y_2^2) - (x_1^2 + y_1^2)\right) \tag{1.6}$$

Když odečteme první rovnici 1.3 i od zbývající rovnice, dostaneme následující matice.

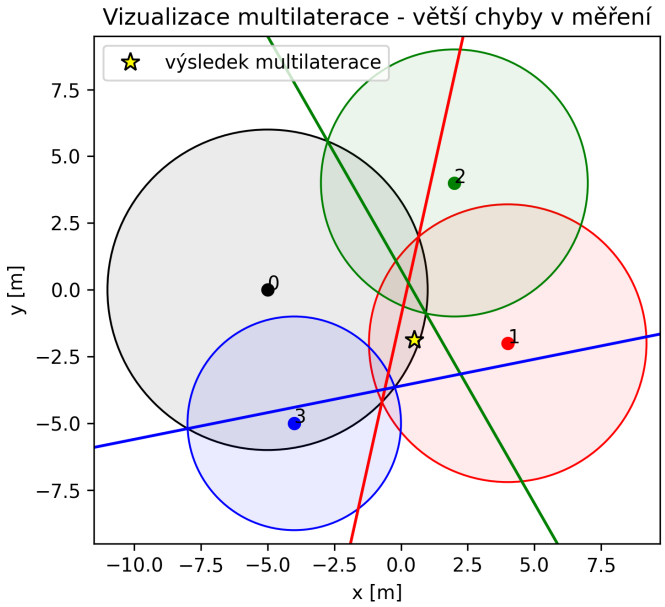


Obrázek 1.6 Ukázkový příklad multilaterace. Kotvy jsou reprezentovány kružnicemi, kde jejich poloměr je naměřená vzdálenost od zařízení. Rovnici kružnice kotvy s číslem 0 odečítáme od ostatních. Přímky reprezentují lineární rovnice multilaterace, které poté řešíme metodou nejmenších čtverců.Rovnice kružnice číslo 0 je odečtena od všech ostatních. Výsledné rovnice jsou vykreslené jako přímky.

$$
\begin{aligned}
A &= \begin{bmatrix} x_2 - x_1 & y_2 - y_1 \\ x_3 - x_1 & y_3 - y_1 \end{bmatrix} \\
b &= \begin{bmatrix} \frac{1}{2}\left(r_1^2 - r_2^2 + (x_2^2 + y_2^2) - (x_1^2 + y_1^2)\right) \\ \frac{1}{2}\left(r_1^2 - r_3^2 + (x_3^2 + y_3^2) - (x_1^2 + y_1^2)\right) \end{bmatrix} \\
x &= \begin{bmatrix} x_0 \\ y_0 \end{bmatrix}
\end{aligned}
\tag{1.7}
$$

$Ax = b$ podle 1.7 nemusí mít řešení, pokud vzdálenosti nejsou přesně změřeny. Tento případ můžeme vidět na obrázku 1.5. Soustavu rovnic vyřešíme pomocí metody nejmenších čtverců, která nám odhadne řešení s nejmenší chybou vůči rovnicím. Existují i lepší způsoby, jak najít řešení soustavy rovnic, ale zvolená metoda dosahuje přijatelných výsledků bez potřeby vysokého výkonu. Obrázky 1.6 a 1.7 zobrazují případy, kdy i z nepřesných měření můžeme získat přijatelná řešení. Výsledné rovnice nám reprezentují přímky procházející průniky s první kružnicí (viz. obrázek 1.6). Případy s ještě větší chybou jsou diskutovány v kapitole 5 společně se srovnáním s výkonově náročnější metodou hledání řešení.

Tento postup můžeme jednoduše rozšířit i pro více zařízení. Přidáním více zařízení se nám přesnost polohy zvětší.



Obrázek 1.7 Ukázka multilaterace v případě, kde jsou velké chyby v měření vzdálenosti.

2 Návrh frameworku

V této kapitole navrhneme framework využívající technologie a návrhy řešení z kapitoly 1.

V první kapitole 1 jsme rozhodli, že budeme využívat Bluetooth mesh na komunikaci a WiFi FTM na měření vzdálenosti. ESP32-S3 zařízení obsahuje pouze jeden RF modul, v jednu dobu může přijímat/vysílat buď pouze WiFi nebo pouze Bluetooth. Podpora pro koexistenci WiFi i Bluetooth na jednom zařízení existuje [16], ale mohlo by to vést k nestabilní komunikaci. Rozdělíme tedy Bluetooth a WiFi komunikaci na dvě zařízení ESP32-S3 tak, aby každé zařízení používalo pouze jednu z technologií.

Každé zařízení může plnit jednu ze dvou rolí, buď je zařízení mobilní nebo je stanicí. Stanice je pevně umístěná (nepohybuje se) a slouží k interakci s uživateli. Mobilní zařízení drží v ruce uživatel a typicky se s ním pohybuje mezi stanicemi.

Jedno zařízení se bude skládat ze dvou modulů, které si rozdělíme na hlavní a vedlejší (také Bluetooth modul). Na hlavním modulu běží samotný framework a uživatelská aplikace. Navíc pak zabezpečuje všechny funkce týkající se WiFi. Tedy hlavně funkce pro měření vzdálenosti a odhad rozložení sítě. Také zde rozlišujeme, zda se jedná o mobilní nebo stacionární zařízení. Vedlejší modul se bude starat o komunikaci se sítí Bluetooth mesh.

2.1 Komunikace mezi moduly

Jelikož jsme ponechali Bluetooth mesh komunikaci na vedlejším modulu, tak musíme nějak komunikovat s hlavním modulem. Výrobce zařízení poskytuje knihovny na komunikaci s vedlejším modulem, ESP-Hosted a ESP-AT. ESP-Hosted je přímo na naše využití, ESP32 modul využitý pouze pro komunikaci s bezdrátovými sítěmi, ale tato knihovna nepodporuje Bluetooth mesh. ESP-AT je knihovna na ovládání ESP modulu pomocí AT příkazů, ale zde narazíme na problém, že Bluetooth mesh také není podporované. Takže si navrhneme vlastní komunikaci.

Pro komunikaci využijeme jednoduchý protokol UART (Universal asynchronous receiver-transmitter). Pomocí tohoto protokolu můžeme posílat jakákoliv data mezi zařízeními. Pro serializaci objektů reprezentující zprávy můžeme využít buď existující knihovny, např. protobuf, nebo si navrhneme vlastní. Mezi moduly nám stačí předávat pouze jednoduchá data, takže nepotřebujeme využívat pokročilé serializace. Navrhneme si jednoduchý protokol založený na textové komunikaci. Toto nám umožní jednoduše vytvořit pomocné programy na jakékoliv platformě, které budou komunikovat s naší sítí Bluetooth mesh.

Před samotným návrhem komunikace si musíme nejprve rozhodnout, jaké informace potřebujeme předávat mezi moduly. Hlavní modul si může buď vyžádat hodnotu vlastnosti zařízení z Bluetooth mesh sítě nebo vlastnost přenastavit na jinou hodnotu. Naopak Bluetooth modul může předávat pouze informace o hodnotách vlastností. Navrhneme tedy textovou komunikaci, kde hlavní modul může pouze zadávat příkazy a Bluetooth modul bude pouze odpovídat posíláním hodnot vlastností zařízení jakmile dostane hodnotu z Bluetooth mesh.

Komunikace může probíhat buď synchronně (po zadání příkazu budeme čekat

na odpověď) nebo asynchronně (bez čekání). Pro komunikaci s vedlejším modulem je jednodušší asynchronní komunikace, jelikož knihovna pro Bluetooth mesh je také asynchronní. To nám navíc umožní i jednoduchou podporu funkce publish v Bluetooth mesh (zařízení posílají stav svého modelu bez vyžádání).

Ještě je nutné definovat, jak se bude framework chovat, když je komunikace asynchronní. Můžeme zavést čekání na odpověď z Bluetooth modulu, tím by se ovšem program zpomalil. Vhodnější řešení je, vytvořit si zvlášť vlákno, které bude odpovědi číst a ukládat je do paměti. Uchovávat si budeme pouze ty nejnovější hodnoty vlastností, které dostaneme od Bluetooth modulu. Když si ve frameworku zažádáme o hodnotu, tak žádost předáme Bluetooth modulu a bez čekání na odpověď vrátíme poslední známou hodnotu této vlastnosti.

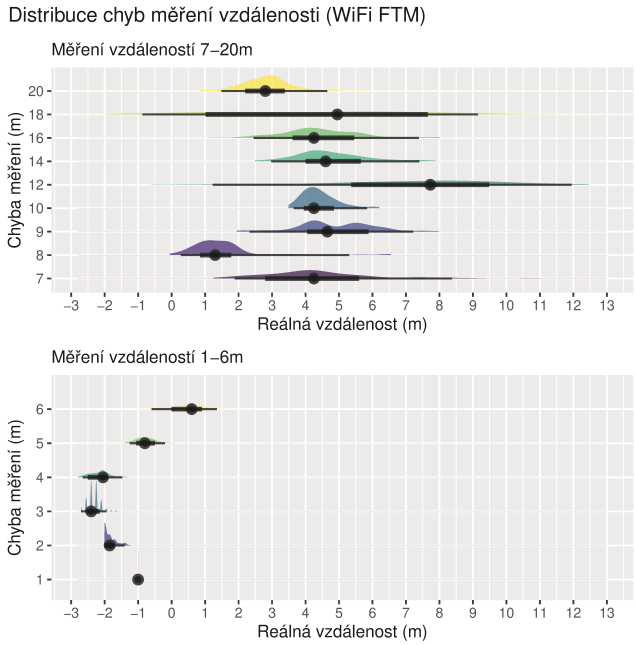
Příkaz se bude skládat z několika částí oddělených mezerou. Jednotlivé příkazy oddělíme znakem new-line (`\n`). První část příkazu bude určovat typ příkazu. Typy budou `GET` a `PUT`, kde pomocí příkazu `GET` si vyžádáme hodnotu vlastnosti a pomocí příkazu `PUT` hodnotu zapíšeme. Potřebujeme rozlišit jednotlivá zařízení a jejich vlastnosti. Zařízení rozlišujeme pomocí jejich adres a vlastnosti si pojmenujeme. Takže druhá část příkazu bude, jakou vlastnost jakého zařízení chceme získat. Adresu oddělíme od vlastnosti pomocí dvojtečky. Pokud chceme komunikovat s lokálním zařízením, tak adresu můžeme vynechat. Např. `0071:rgb` označuje vlastnost `rgb` pro zařízení s adresou `0x0071`. Navíc pro `PUT` potřebujeme předat novou hodnotu. Takže pro tento typ příkazu přidáme třetí část, kde předáme hodnotu.

Bluetooth modul bude jako odpověď posílat pouze hodnoty vlastností zařízení. Označení zařízení a jeho vlastnosti necháme stejné jako u příkazu. Za to přidáme znak rovná se (`=`) a hodnotu vlastnosti. Např. `0071:rgb=ffffff` znamená, že `rgb` vlastnost zařízení s adresou `0071` má hodnotu `ffffff`. Jednotlivé zprávy zase oddělíme znakem new-line (`\n`).

2.2 Měření vzdálenosti

Měřit vzdálenosti budeme pomocí WiFi FTM. Pro měření vzdálenosti je zapotřebí WiFi AP s podporou FTM Responder a WiFi klient s FTM Initiator. Samotné měření poté probíhá na stejném kanálu WiFi AP. Naše zařízení podporují obě role a dokonce i souběžný běh AP a klienta pod podmínkou, že komunikují na stejném kanálu. Tedy všechny zařízení musíme umístit na stejný komunikační kanál pro zajištění podpory souběžného zahájení měření vzdálenosti i poskytování měření vzdálenosti ostatním zařízením. To z hlediska výkonu není doporučené, jelikož komunikace více zařízení na stejném kanálu vede k vzájemnému rušení a tím se snižuje celkový výkon sítě. Omezíme tedy počet zařízení, která vysílají na stejném kanálu. Mobilní zařízení i stanice potřebují v průběhu aplikace měřit vzdálenost k ostatním. Zaměříme se tedy na omezení počtu AP. Stanice potřebují vytvořit AP, jelikož slouží jako kotvy v lokalizačním systému. Mobilním zařízení stačí pouze měřit vzdálenosti ke stanicím pro určení své polohy. Pokud by aplikace potřebovala měřit vzdálenost mezi mobilními zařízení, tak si vzdálenost může odhadnout pomocí určené polohy ostatních mobilních zařízení. Aplikace tedy bude umožňovat měřit vzdálenosti pouze ke stanicím (mobilní→stanice, stanice↔stanice).

Z experimentu měření vzdálenosti jsme zjistili, že rozptyl chyby v měření je velký, hlavně u větších vzdáleností (viz. obrázek 2.1). Pro zmenšení rozptylu



Obrázek 2.1 Graf distribuce chyby při měření vzdálenosti pomocí WiFi FTM na zařízeních ESP32-S3. Měření bylo provedeno ve venkovním prostředí. Pro každou reálnou vzdálenost bylo provedeno 80–100 měření, Měření vzdálenosti není přesné. Například u vzdálenosti 12 metrů je chyba hodně rozptýlená, ale u 3 metrů je rozptyl chyby malý. To může být dané prostředím, rádiové signály se mohou odrážet od různých povrchů a to ovlivňuje měření.

budeme měření filtrovat. Pro zmenšení rozptylu je dostatečné provést průměr z více naměřených hodnot.

Chování filtrování upravíme podle typu zařízení. Stanice měří vzdálenosti pouze k ostatním stanicím. Můžeme tedy provést průměr z velkého počtu měření (např. 100), jelikož se reálné vzdálenosti se mezi nimi nemění. Mobilní zařízení musí rychle reagovat na změnu vzdálenosti, jelikož se mohou pohybovat. Počet průměrovaných hodnot musíme tedy snížit na nejmenší přijatelnou hodnotu nebo nejlépe neprovádět průměr vůbec.

2.3 Lokalizační systém

Lokalizační systém slouží ve frameworku k určení polohy zařízení v rámci lokálního souřadnicového systému definovaného rozmístěním stanic.

Lokalizační systém rozdělíme na dvě části. V první části určíme polohy stanic a ve druhé části budeme určovat polohy mobilních zařízeních. Princip určení polohy je u obou částí stejný, ale liší se postupem. Pro jednoduchost budeme nazývat stanice, které mají určenou polohu a můžeme si k nim zjistit vzdálenost, *kotvami*.

Začneme určením polohy pro mobilní zařízení. Předpokládejme, že známe polohy stanic a vzdálenosti k nim. Poté pokud jsou dostupné alespoň 3 kotvy, tak si můžeme pomocí multilaterace vypočítat polohu, ze které proběhlo měření vzdáleností.

Pro stanice použijeme stejný způsob určení polohy. Stačí nám určit polohy prvních tří zařízení. Poté stačí zajistit, aby každá stanice měla alespoň 3 kotvy, a

polohu si může stanice určit stejně jako mobilní zařízení.

První tři zařízení si určí polohu následovně. Předpokládáme, že zařízení si mohou změřit vzdálenost mezi sebou. První zařízení nám určí počátek souřadnicového systému, polohu tedy nastavíme na (0,0). Druhé zařízení může být kdekoliv na kružnici se středem s polohou prvního zařízení a poloměrem d, kde d značí vzdálenost mezí prvním a druhým zařízením. Umístíme jej na souřadnice $(d, 0)$, tím si určíme směr osy x. Třetí zařízení nesmí být na stejné přímce jako první dvě. Pokud to splníme, tak jej umístíme pomocí průniku dvou kružnic, první kružnice bude reprezentovat vzdálenost k prvnímu bodu a druhá reprezentuje vzdálenost k bodu druhému. Dostaneme dva průniky [17], můžeme si pak vybrat jaký z nich použijeme jako pozici třetího zařízení. Vybereme tedy průnik, který vlevo od osy x. Výběrem průniku si tedy určíme směr osy y. Pokud směr osy y nebude odpovídat reálnému rozložení stanic, tak to lokalizaci nijak neovlivní, pouze to může vadit uživateli při vizualizaci pozic zařízení.

Pokud je dostupných více kotev než je nutné na určení polohy, tak je dobré si vybrat, jaké budeme používat na určení polohy. I jedna špatně změřená vzdálenost může mít velký vliv na výslednou polohu. Pokud jsou dvě zařízení blízko u sebe a signál mezi nimi je blokován, tak naměřená vzdálenost může být vysoká, jelikož signál se může odrazit od vzdáleného objektu. Přímá vzdálenost mezi zařízeními je vždy ta nejkratší. Pokud si vezmeme pouze 5 nejbližších kotev, předejdeme takto velkým chybám v lokalizaci. Soustava rovnic pro výpočet polohy bude mít pořád více rovnic, než je nutné, takže tolerance na menší chyby bude zachována.

Nevýhoda tohoto postupu lokalizace kotev je, že se chyby jednotlivých výsledků lokalizace akumulují. Každá určená poloha pomocí této metody, zahrnuje v sobě chyby poloh zařízení, které využívá k lokalizaci, a chyby naměřených vzdáleností k nim. V menších sítích výsledky této akumulace nepoznáme, ale ve větších sítích můžou být chyby zásadní. Pro zajištění přesnosti lokalizace mobilních zařízení umožníme provozovateli sítě sledovat umisťování stanic a případně opravit vzniklé chyby ručně.

2.4 Bluetooth mesh

Pomocí Bluetooth mesh, jenž bude součástí vedlejších modulů, budeme řešit komunikaci mezi zařízeními, a to jak stanic, tak mobilních zařízení. Framework bude komunikovat s Bluetooth mesh pomocí sériové komunikace navržené v kapitole 2.1.

Všechny zařízení v rámci Bluetooth mesh budou poskytovat komunikaci barvy, polohy, stavu aplikace a zapnutí/vypnutí světla. V následujícím textu vybereme modely pro komunikaci jednotlivých vlastností a navrhneme struktury zařízení Bluetooth mesh.

Barva světla Hlavní způsob komunikace s uživateli bude pomocí barvy světla. Bluetooth mesh modely obsahují několik modelů na ovládání barvy světla. U barevných LED se jejich barva ovládá většinou jako poměr základních barev — červené, zelené a modré. Bluetooth mesh neobsahuje žádný takový model. Místo toho jsou světelné modely založené na jiných barevných prostorech, které více souvisí s vnímáním barev u lidí. Konkrétně HSL a xyL. Oba modely mají 3 čísla,

Element 1	Element 2	Element 3
Configuration server	RGB element 1 server	RGB element 2 server
Health server		
Generic OnOff server		
Generic OnOff klient		
RGB server		
RGB setup server		
RGB klient		
Generic Location server		
Generic Location setup server		
Generic Location klient		
Generic Level server		
Generic Level klient		

Tabulka 2.1 Tabulka popisuje strukturu elementů Bluetooth mesh. Hlavní používané modely jsou zvýrazněné. Ostatní modely jsou hlavně na splnění požadavků Bluetooth mesh. RGB modely jsou pouze abstrakce nad HSL modely vypracované v této práci.

které udávají výsledný stav barvy světla. Poslední číslo je u obou modelů stejné, a to L=*perceived* ***l****ightness* neboli jas barvy podle vnímání lidského oka. U HSL modelu se udává odstín (**h**ue) a saturace barvy (**s**aturation). U xyL se barva udává pomocí souřadnic x a y v barevném prostoru CIE1931. Pro převod do RGB hodnot je jednodušší model HSL, tedy jej v této práci zvolíme. Nad HSL modelem vypracujeme abstrakci, která bude převádět RGB hodnoty na HSL a naopak.

Poloha Pro komunikaci polohy máme na výběr ze dvou modelů, které reprezentují buď lokální nebo globální souřadnice. Lokalizační systém navržený v kapitole 2.3 využívá lokální souřadnice, zvolíme tedy model *Generic Local location*. Lokální poloha se v modelu reprezentuje jako 3 celočíselné hodnoty, které udávají jako pozice *north*, *east* a *altitude* v rámci lokálního systému souřadnic. Navíc model obsahuje položky označující patro (*floor number*) a nepřesnost polohy (*uncertainty*).

Stav aplikace Výsledná aplikace by měla mít možnost měnit své chování podle stavu zařízení. Pro komunikaci stavu nám poslouží model *Generic Level*, který předává celočíselné číslo. Model umožňuje i více věcí, např. plynulý přechod mezi dvěma čísly nebo relativní změna čísla, které nebudeme využívat.

Zapnutí/vypnutí světla Pro zapnutí/vypnutí světla poskytuje Bluetooth mesh model *Generic OnOff model*. Model kromě jednoduchého vypínače poskytuje možnost nastavit plynulý přechod mezi stavy, tuto funkčnost využívat nebudeme.

Celou strukturu elementů můžeme vidět v tabulce 2.1. Pro využití HSL modelu potřebujeme mít 3 elementy, jelikož se vyžaduje mít zvlášť modely na ovládání odstínu a saturace bez ovlivnění ostatních hodnot, jinak bychom mohli mít pouze jeden element.

Stav	Barva	Stanice	Mobilní zařízení
0 — `INIT`	černá	nic	nic
1 — `DM`	červená	měření vzdáleností	nic
2 — `LOC`	růžová	lokalizace	nic
3 — `APP`	zelená	běh aplikace	měření vzdáleností +lokalizace +běh aplikace

Tabulka 2.2 Tabulka popisuje základní stavy frameworku. Funkčnost stavů se liší podle typu zařízení, na kterém běží framework. Barva stavu je zobrazená na LED hlavního modulu (černá = nesvítí).

2.5 Stavy aplikace

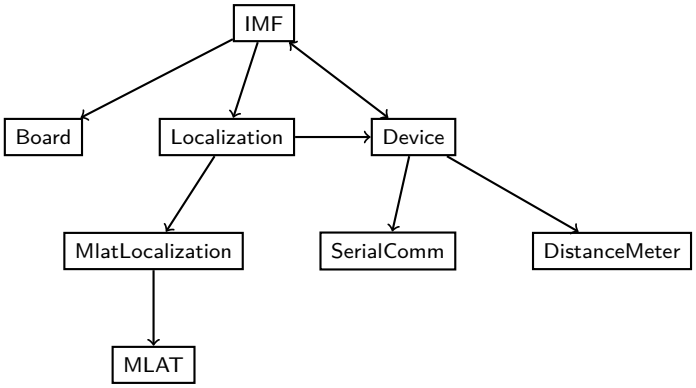
Framework podporuje definici různých stavů a přepíná mezi nimi. Každý stav má přidělenou jednu funkci, kterou framework pravidelně volá. Pokud nespecifikujeme jinak, tak si framework vytvoří vlastní stavy (viz. tabulka 2.2). Číslování základních stavů je navrženo tak, aby číslo stavu pro běh výsledné aplikace byl stejný na mobilním zařízení i stanici. Po zapnutí zařízení se nastaví stav na 0.

2.6 Ostatní funkce

Persistentní logování ESP-IDF poskytuje základní funkce na logování [18], ale neposkytuje ukládání logů. Ukládání logů je pro mobilní zařízení celkem důležité, protože pokud bychom měli nějakou vzácnou chybu v aplikaci, tak bychom o ni nemuseli vůbec vědět. Mít pořád připojený počítač k mobilním zařízením není vůbec praktické, někdy i nemožné. V tom nám může pomoct persistentní logování. Můžeme si nechat zaznamenat veškeré logovací zprávy a mít je uložené na zařízení a jednou za čas si logy projít a podívat se, jestli nám aplikace nehlásí nějaké chyby. Vytvoříme si tedy vlastní logovací knihovnu, která bude podporovat ukládání logů do zařízení.

Konfigurace zařízení Aplikace většinou umožňují přenastavit chování programu, podle našich preferencí. Pro mikrokontroléry se takové nastavení častokrát řeší při kompilaci programu, jelikož ve výsledku bude aplikace nejrychlejší a bude zabírat nejméně paměti. To je pro typické nasazení mikrokontrolérů nejlepší řešení. Pokud s aplikací interaguje uživatel, tak je dobré mít jednoduché nastavení, které můžeme měnit bez kompilace programu. Součástí tohoto frameworku je tedy vypracován modul na konfiguraci zařízení s webovým uživatelským rozhraní i API. Aplikace si může přidat záznamy do této konfigurace a přečíst si uloženou hodnotu. Konfigurace se udržuje i po restartovaní zařízení nebo výpadku napájení.

Konfigurační modul je vytvořený tak, že pro možnost úprav nastavení se aplikace přepne do konfiguračního režimu. Pokud chceme vystoupit z konfiguračního režimu, tak zařízení restartujeme. Způsob přepnutí do konfiguračního režimu je přenecháno na výsledné aplikaci.



Obrázek 2.2 Struktura hlavních tříd projektové implementace. Šipky znázorňují závislosti jednotlivých tříd na sobě.

2.7 Pomocná aplikace

Pro jednodušší práci s frameworkem byla vyvinuta aplikace na počítač, která nám zjednoduší práci s obsluhou výsledné aplikace. Se zařízeními komunikujeme pomocí Bluetooth mesh protokolu. Pro připojení počítače k síti využijeme stejný Bluetooth modul jako framework. Modul připojíme pomocí USB–UART adaptéru k počítači a budeme s ním komunikovat stejně jako hlavní modul.

Hlavní funkcí aplikace bude vizualizace sítě. Při rozmisťování zařízení si totiž nemůžeme ze zařízení jednoduše zjistit, jestli se poloha zařízení určila správně. Aplikace si tedy zjistí polohy zařízení a graficky je zobrazí. Umožníme také uživateli případně opravit pozice zařízení. Pro uživatelskou přívětivost se polohy budou upravovat přesunutím grafické reprezentace zařízení pomocí myši na novou polohu.

Dále umožníme uživateli upravovat i ostatní vlastnosti zařízení, ale pouze ty, které jsou dostupné přes Bluetooth mesh. Do aplikace dáme grafické rozhraní pro formulaci příkazu, který se pošle Bluetooth modulu. Uživatel bude moct poslat zprávu i všem zařízením najednou pomocí zvolení správné adresy.

2.8 Technická implementace

V této kapitole se budeme zabývat technickými detaily implementace frameworku. Na obrázku 2.2 můžeme vidět jak jsou hlavní třídy propojeny mezi sebou.

Framework je programovaný za pomocí frameworku ESP-IDF v5.1 od výrobce zařízení v jazycích C a C++. Práce je rozdělena do komponent, které lze použít i samostatně v jiném projektu. Hlavní komponenta je `interactive-mesh-framework`, která spojuje jednotlivé komponenty a stará se o inicializaci zařízení.

2.8.1 Komunikace mezi moduly

Komunikace je rozdělená na dvě části, server a klient. Klient si vyžádá data ze serveru a server je asynchronně pošle klientovi.

Komunikace je rozdělena do tří tříd. Abstraktní třída `SerialComm` a její potomci `SerialCommSrv` a `SerialCommCli`.

Třída `SerialComm` implementuje formátování zpráv a odpovědí a samotnou komunikaci přes UART. Zpráva se skládá z názvu pole a případně nové hodnoty. Pro zahájení spuštění čtení odpovědí zavoláme metodu `startReadTask`, která spustí asynchronní čtení.

Třídy `SerialCommSrv` a `SerialCommCli` jsou potomci `SerialComm` a definují chování klienta a serveru. Implementují zpracování odpovědí a formátování názvu polí (do názvu pole přidají adresu zařízení). Pro zpracování odpovědí implementujeme virtuální funkci `processInput`, kde jako parameter je přijatý řetězec reprezentující odpověď. Metoda zabezpečuje i případná řešení špatně formátovaných odpovědí.

Při zpracování odpovědi si klient ověří správnost formátování a rozdělí si odpověď na jednotlivé části, které se uloží do paměti společně s časem přijmu.

Jelikož se Bluetooth mesh může jednoduše zahltit velkým počtem zpráv, klient omezuje počet zpráv poslaných Bluetooth modulu. Při žádosti aplikace o hodnotu vlastnosti zařízení budeme žádost předávat Bluetooth modulu pouze v případě, že poslední čas aktualizace hodnoty přesáhne předem definovaný práh. Tímto mechanismem můžeme také předejít posíláním `GET` požadavků Bluetooth modulu, pokud si u každého zařízení nastavíme pravidelné 'publish' na adresu `ffff` (všechna zařízení) u všech modelů.

2.8.2 Lokalizační systém - multilaterace

Lokalizační systém dostane seznam všech stanic. Pro určení polohy si musíme nejprve zjistit, které stanice použijeme jako kotvy k určení polohy. Nejprve vyřadíme stanice, ke kterým nedokážeme zjistit vzdálenost, poté vyřadíme zařízení, která nemají ještě určenou polohu. To poznáme podle parametru nepřesnosti v poloze. Při spuštění zařízení nastavíme nepřesnost na maximální hodnotu (65535). Určíme si práh nepřesnosti, podle kterého budeme filtrovat stanice, konkrétně 65535/2.

Zařízení měří vzdálenost v centimetrech. Pro jednoduchost převedeme vzdálenost do metrů.

Jakmile dokončíme seznam kotev, tak se podle počtu rozhodneme, jak budeme určovat polohu zařízení. Pokud nejsou žádné kotvy, tak polohu nastavíme na (0,0). Pokud je k dispozici pouze jedna kotva, tak poloha bude $(x_0,y_0 + d)$, kde (x_0,y_0) je poloha kotvy a d je naměřená vzdálenost od ní. Pro dvě kotvy bude poloha zařízení jeden z průniků kružnic reprezentující kotvy. Pro tři a více kotev polohu určíme pomocí multilaterace.

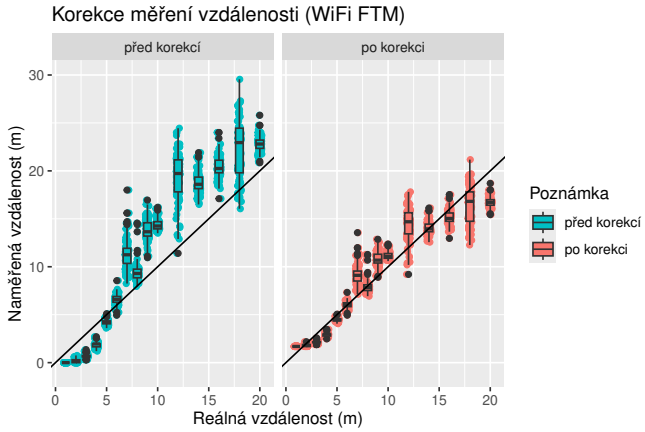
Multilaterace je počítána za pomocí floating-point čísel pro dosažení nejvyšší přesnosti. Pro převod polohy zařízení do Bluetooth mesh, kde jsou pozice uloženy celočíselně, si vypočítané polohy přeškálujeme, abychom si zachovali přesnost a zároveň využili možný rozsah poloh. Při zachování dvou desetinných míst budeme mít možný rozsah poloh -327.68–327.67 m.

2.8.3 Hlavní třída

Hlavní třída `IMF` se stará o celý chod aplikace. V konstruktoru této třídy inicializuje vše potřebné pro chod aplikace na našem zařízením. Výsledná aplikace, která využívá náš framework, třídy inicializuje a nastaví podle svých potřeb a

Program 1 Příklad použití metody objektu jako callback funkci pro stav číslo 6.

```
auto state_cb = [this](TickType_t diff) { this->method(); };
imf.setState(6, state_cb, (rgb_t){255,255,255});
```



Obrázek 2.3 Srovnání měření vzdálenosti před a po korekci pomocí lineární regrese rovnice $y = x + b$, kde y je reálná vzdálenost a x je naměřená vzdálenost. Měření vzdálenosti bylo provedeno pomocí WiFi FTM na zařízeních ESP32-S3. Měření bylo provedeno ve venkovním prostředí. Pro každou reálnou vzdálenost bylo provedeno 80–100 měření. Výsledek lineární regrese je $y = 0.659338 \cdot x + 1.687536$. Z grafu vidíme, že po korekci jsou méně rozptýlená a jsou blíže diagonále, která reprezentuje přesné měření. Po korekci nemůžeme naměřit méně než 1.68 metrů. Pokud s tím aplikace počítá, tak nám to nevadí, protože to zpřesní ostatní měření.

poté předá kontrolu frameworku. Aplikace následně provádět kód pouze v callback funkcích. Nejdůležitější callback funkce pro aplikace jsou pravidelný `update` a zmáčknutí tlačítka. Callback funkce je potřeba na začátku programu registrovat, jinak aplikace ztratí možnost provádět kód.

Další funkčností třídy je stav aplikace. Podle stavu aplikace bude pravidelně provádět jiné funkce. Stav aplikace je daný celočíselnou hodnotou `level`. Třída `IMF` umožňuje kromě přednastavených stavů si nadefinovat i vlastní stavy s vlastními callback funkcemi. Můžeme si i zaregistrovat callback na změnu stavů.

Stav má k sobě přiřazenou callback funkci, která se pravidelně volá. Funkce musí být typu `std::function<void(TickType_t)>`. Pokud chceme jako callback dát metodu třídy, tak můžeme vytvořit lambda funkci se zachycením kontextu objektu se stejným předpisem funkce (viz. program 1).

2.8.4 Měření vzdálenosti

Měření vzdálenosti je implementováno jako samostatná třída `DistanceMeter`. Samotné měření vzdálenosti probíhá v pomocné třídě `DistancePoint`, která reprezentuje bod, ke kterému můžeme měřit vzdálenost.

Měření vzdálenosti ve třídě `DistancePoint` probíhá následovně. Zahájí se měření vzdálenosti pomocí WiFi FTM a program čeká než se měření provede. Výsledek měření dostaneme v eventu `WIFI_EVENT_FTM_REPORT`, ten si zkopírujeme do statické globální proměnné a oznámíme to čekajícím vláknům. Výsledkem

měření může být i neúspěch měření. Jakmile se čekající vlákno probudí, tak si převezme odhadovanou vzdálenost. Na tuto vzdálenost provedeme korekci, kterou získáme z experimentálního měření, např. pomocí lineární regrese (viz. obrázek 2.3). Poté výslednou vzdálenost dáme do filtrovacího algoritmu, v našem případě to je klouzavý průměr. Pokud nemáme dostatek hodnot pro naplnění jednoho okna v klouzavém průměru, tak provedeme normální průměr na naměřených datech. Vzdálenost si nakonec uložíme.

Třída `DistanceMeter` zabezpečuje udržování jednotlivých bodů a pravidelné měření vzdálenosti. Třída také zjišťuje jestli jsme v blízkosti nějakého bodu. Tuto informaci předává výsledné aplikaci pomocí eventu `DM_NEAREST_DEVICE_CHANGE`. V daný okamžik můžeme být v blízkosti pouze jednoho bodu. Pro označení, že jsme v blízkosti nějakého bodu, musí být vzdálenost k tomuto bodu nejmenší a navíc musí být vzdálenost menší než 10 m. Abychom zohlednili i čas měření v porovnávání vzdáleností, tak budeme předpokládat, že od posledního času naměření se vzdalujeme od bodu konstantní rychlostí.

Poté je zde event `DM_MEASUREMENT_DONE`, který se vytvoří po každém měření vzdálenosti.

`DistancePoint` si uchovává posledních 5 výsledných vzdáleností. Tyto vzdálenosti si můžeme získat pomocí metody `getDistanceFromLog` společně s časem měření.

2.8.5 Webová konfigurace

Komponenta `web_config` se stará o konfiguraci zařízení přes webové rozhraní.

Jakmile se zařízení přepne do konfiguračního režimu, tak se buď připojí k existující WiFi AP nebo si vytvoří vlastní. Rozhoduje se následovně: nejprve se zkusí připojit na WiFi AP nastavené ve webové konfiguraci, poté zkouší WiFi AP nastavené při kompilaci programu. Pokud se nepřipojí na žádnou WiFi, tak si vytvoří vlastní WiFi. Pokud jsou nastavené údaje v konfiguraci, tak je použije. Nakonec, pokud se nepodaří ani jedna z předchozích možností, tak si založí WiFi s názvem `NODE-XXXXXXXXXXXX`, kde `X` nahradíme MAC adresou zařízení s převráceným pořadím bytů[1].

Webová konfigurace poskytuje dvě možnosti přístupu, buď přes webové rozhraní, které je na adrese `<ip-adresa>/`, nebo přes API, které je na adrese `<ip-adresa>/api/v1/`.

- `/api/v1/nvs/<key>`, kde `<key>` je název nastavení (i se znaky `/`)
  - GET request vrátí hodnotu nastavení
  - POST request nastaví hodnotu nastavení na hodnotu předanou v těle requestu
- `/api/v1/nvs_save` — GET request uloží změněná nastavení do paměti zařízení (pokud se nezavolá, tak není zaručené uložení změn)
- `/api/v1/logs`

[1] Přehozené pořadí bytů nám umožní rychlejší rozlišení různých zařízení při čtení názvu WiFi zleva do prava.

  - GET request vrátí veškerý obsah log souboru
  - DELETE request vymaže log soubor a vytvoří nový
- `/api/v1/reboot` — GET request restartuje zařízení
- `/api/v1/ota` — POST request začne OTA aktualizaci firmwaru z adresy, které je v těle requestu

Aplikace může přidat vlastní nastavení do webové konfigurace pomocí metody `addOption` třídy IMF. Pro přidání nastavení definujeme název nastavení (slouží jako klíč nastavení), typ hodnoty, validační funkci a parametr zda hodnota může být zapsána do logu. Hodnota nastavení může být typu, buď string (`NVS_TYPE_STR`), nebo 16 bitové číslo signed (`NVS_TYPE_I16`) nebo unsigned (`NVS_TYPE_U16`). Validační funkce slouží pro kontrolu správnosti hodnoty při ukládání nastavení pomocí webové konfigurace.

Číst a zapisovat nastavení může i aplikace. Nastavení se ukládá do NVS (Non-Volatile Storage) paměti. Číst nastavení můžeme pomocí funkcí `nvs_get_*` a zapisovat můžeme pomocí funkcí `nvs_set_*`, kde místo `*` bude `str`, `i16` nebo `u16` podle typu hodnoty. Pro přístup k nastavení potřebujeme `nvs_handle_t`, který můžeme získat pomocí metody `getOptionsHandle` třídy IMF. Podrobnosti čtení a zápisu do NVS jsou v dokumentaci ESP-IDF [19].

2.8.6 Persistentní logování

Logovací knihovna podporuje více výstupů, na standardní výstup, přes UART a do souboru. Při inicializaci logování si můžeme zvolit jaké výstupy požadujeme.

Výstup do souboru Soubor ukládáme do vnitřní paměti modulu. Tato paměť má omezený počet zápisů a je typicky malá, v této práci používáme modul s kapacitou 16 MB. Takže bychom měli zapisovat do logu pouze nejdůležitější data. Knihovna počítá s tím, že logovací soubor může být větší než je paměť na modulu. Pokud velikost souboru přesáhne předem definovaný práh, tak se obsah souboru začne přepisovat od začátku. Při inicializaci logu si můžeme zvolit, kolik počátečních bytů nemůžeme přepsat. Knihovna se tomuto regionu bude vyhýbat při zapisování do souboru, aby ochránila případné chybové hlášky, které nastanou krátce po spuštění programu.

Paměť modulu se při kompilaci rozděluje na části. Každá část slouží na jiný účel. Pro ukládání logů si vytvoříme partition s názvem `logs` a typem `SPIFFS` (v souboru `partitions.csv`). Jako souborový systém využíváme `LittleFS`. Je to speciální souborový systém vytvořený pro mikrokontroléry. Je přizpůsobený i případným náhlým výpadkům napájení.

Logovací soubor si můžeme pojmenovat dle potřeby. Nicméně pokud je zapotřebí využívat webové konfigurace k přístupu k logům, tak soubor musí mít výchozí název `/logs/log.txt`

2.8.7 Pomocná aplikace

Pomocná aplikace byla vyvinutá v jazyce Python pro jednoduché ovládání zařízení. Pro grafické rozhraní i komunikaci se zařízením používá aplikace knihovnu PySide6.

PySide6 je založená na populární knihovně Qt. Poskytuje multiplatformní grafické rozhraní pro všechny hlavní operační systémy na PC, Windows, MacOs i Linux[2].

Aplikace vychází z oficiálních příkladových programů na vizualizaci grafů z knihovny NetworkX a terminálem pro komunikace přes sériový port. Grafické rozhraní aplikace se skládá ze dvou částí. Hlavní část programu je vizualizace rozmístění zařízení. Dále aplikace obsahuje grafické rozhraní pro změnu vlastností zařízení dostupných z Bluetooth mesh sítě.

Vizualizaci rozmístění zařízení zabezpečuje třída *GraphView*, která je potomek třídy *QGraphicsView*, které nám poskytuje plochu pro umístění obrázků. Třída zabezpečuje i pravidelné získávání informací o zařízeních a správu grafické reprezentace zařízení. Aplikace umožňuje převrácení kolem osy x i y pomocí tlačítek *flip x* a *flip y*. Tyto tlačítka mají vliv pouze na zobrazení, pozice na zařízeních se nezmění.

Každé zařízení je graficky reprezentované pomocí třídy *NodeG*, která je potomkem *QGraphicsObject*. Objektu povolíme přesouvání pomocí myši nebo dotykové obrazovky. Objekt obsahuje vlastní vykreslovací funkci, která vykreslí obdélník s textovou reprezentací objektu zařízení. Pozice objektu nám udává střed obdélníku.

Součástí *GraphView* je časovač, který každých 5 s spustí funkci na aktualizaci. V rámci aktualizace odešleme Bluetooth modulu požadavek o nové hodnoty pozic zařízení. Navíc zkontrolujeme, zda uživatel nepřemístil zařízení v grafické reprezentaci. Pokud ano, tak odešleme příkaz Bluetooth modulu na změnu pozice daného zařízení. Komunikace funguje asynchronně s Bluetooth modulem. Takže jakmile dostaneme nové informace od modulu, tak to oznámíme objektu *GraphView* a ten aktualizuje zobrazení. Při aktualizaci polohy upřednostňuje program změnu polohy od uživatele před novou polohou zařízení. Takže pokud dostaneme informaci o nové poloze zařízení a zařízení není v grafické reprezentaci na poslední poloze přiřazené programem, tak odešleme příkaz o změně poloze. To se děje do té doby, než uživatelem změněná poloha není stejná jako poloha zařízení.

[2]Aplikace byla vyvíjena a testována v Linuxovém prostředí, konkrétně Fedora 39.

3 Zařízení

V této kapitole se budeme zabývat zařízeními, na kterých bude framework testován.

V rámci práce byly navrženy a vyrobeny dva typy zařízení. Jsou založeny na modulu ESP32-S3-WROOM-1, který obsahuje ESP32-S3 čip, paměť, anténu a všechny podpůrné obvody. Pro zprovoznění tohoto modulu stačí připojit USB konektor, napájení (3.3 V) a správně zapojit piny.

První typ zařízení je podobný vývojové desce. Kromě základních součástek deska připojuje LED na pin GPIO38 a tlačítka na restartování čipu a pro bootloader režim. Ostatní piny jsou vyvedené na kraj desky. Tato deska slouží na jednoduchý vývoj nových zařízení.

Druhý typ zařízení je specializovaný pro naší aplikaci. Piny jsou seskupené tak, aby se jednoduše připojily přídavné desky. Oproti prvnímu typu jsou zde konektory na desku pro uživatelský vstup/výstup, displej, přídavný modul a microSD kartu.

Schéma obou typů zařízení a navržené tištěné spoje (ukázka na obrázku 3.1) jsou součástí přílohy ve složce `boards/`.

Pro naší aplikaci budeme využívat oba typy zařízení. Specializovaný typ využijeme jako náš hlavní modul, kde poběží framework, a druhý typ využijeme jako komunikační modul Bluetooth mesh. Zařízení propojíme podle tabulky 3.1 nebo podle obrázku 3.1.

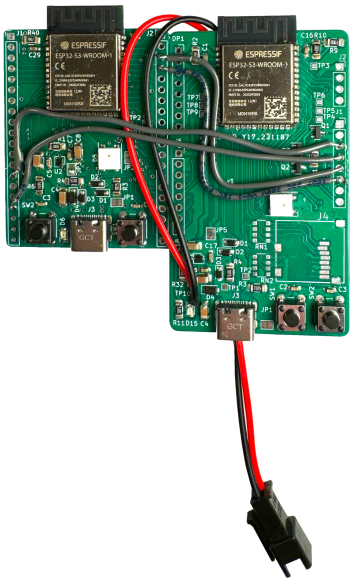
Zařízení je napájeno pomocí jedné 18650 Li-Ion baterie. Pro základní ochranu baterie byli pořízeny ochranné moduly kombinované s nabíjecím obvodem, konkrétně „Hailege 4056 USB Type-C 5V 1A Lithium Battery Charger Module Charging Board with Dual Protection Functions“. Zapojení můžeme vidět na obrázku 3.2. Pokud výstup zkratujeme, tak tento obvod baterii odpojí, aby nedošlo k jejímu poškození. Ochranný modul je kombinovaný s nabíječkou baterie přes USB-C konektor[1]. Během této práce byly baterie nabíjeny externí nabíječkou.

Když je baterie zapojená, jak je zobrazeno na obrázcích 3.1 a 3.2, tak nesmíme zařízení s připojenou baterií zapojit přes USB ke zdroji napětí, např. k PC. Baterie by se mohla poškodit, protože by byla připojená k vyššímu napětí než je doporučeno. Ochranný modul by baterii měl ochránit, nicméně není vhodné na to spoléhat. Pokud bychom to chtěli, tak přidáním diody mezí červený drát z ochranného modulu a zařízením zamezíme vedení proudu z USB portu do baterie.

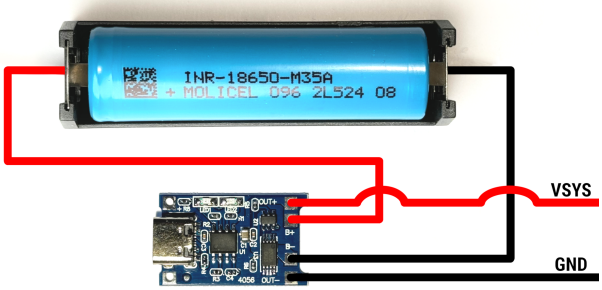
[1]Nabíjecí obvod není vhodný, jelikož USB-C konektor je špatně zapojený, CC1 a CC2 piny nejsou nikam připojeny. Takže nebude fungovat s kabelem USB-C na USB-C. Navíc se nabíjecí čip při nabíjení hodně zahřívá.

Hlavní zařízení (a)	Bluetooth zařízení (b)	Funkce
GPIO41	GPIO42	komunikace (a)←(b)
GPIO42	GPIO41	komunikace (a)→(b)
3V3	3V3	napájení
GND	GND	země

Tabulka 3.1 Tabulka propojení modulů.



Obrázek 3.1 Obrázek zapojení zařízení vyrobených pro tuto práci. Vlevo je zařízení `esp32-s3-simple`, který zabezpečuje komunikaci pomocí Bluetooth mesh. Vpravo je modul `esp32-s3-board`, který zabezpečuje běh frameworku. K hlavnímu modulu je připojený kabel s koncovkou JST SM 2P pro připojení baterie.



Obrázek 3.2 Obrázek zapojení baterie pro napájení zařízení. Nahoře je 18650 baterie v držáku. Dole je ochranný modul s nabíječkou.

4 Ukázková aplikace

V této kapitole si předvedeme použití frameworku na hře „Najdi svoji barvu“. Na předem definované ploše jsou rozmístěny stanice s určitou barvou. Jednotlivý hráči dostanou mobilní zařízení a cílem hry je donést mobilní zařízení ke stanici, na které svítí stejná barva, jako je mobilním zařízení hráče.

Mobilní zařízení a stanice mají odlišné chování v této hře, tedy si aplikaci rozdělíme na dva projekty. V přílohách je dostupný postup na vytvoření nového projektu A.2, nahrání kódu na zařízení A.1. Začneme jednodušší aplikací, a to aplikací pro stanici, a poté se podíváme na aplikace pro mobilní zařízení, která je už složitější. Uživatelskou dokumentaci k aplikace najdeme v příloze A.3.

4.1 Stanice

Stanice v naší hře jsou pasivní. Po určení své polohy jsou využívané hlavně k odpovídání na měření vzdálenosti od jiných zařízení. Tedy si na této aplikaci ukážeme inicializaci frameworku.

Pro zprovoznění frameworku je zapotřebí:

- vytvořit si objekt třídy `IMF`, nejlépe aby byl globálně dostupný pro naší aplikaci
- přidáme si informace o našich zařízeních pomocí metody `addDevice`
- zaregistrujeme si callback funkce, pokud některé nechceme, tak předáme `nullptr` místo názvu lokální funkce
- nakonec stačí zavolat metodu `start` a framework převezme kontrolu

Minimální program pro spuštění frameworku můžeme vidět v programu 2.

Potřebujeme si udělat mapování mezi MAC adresy hlavních modulů zařízení a Bluetooth mesh adresy jejich vedlejších modulů. Navíc zadáme kanál, na kterém budou mít WiFi AP, nejlépe budou všechny zařízení na stejném kanálu. Nakonec přidáme informaci o tom, jestli je zařízení mobilní, nebo je stanice. Tyto informace předáme frameworku pomocí metody `addDevice`.

Pro stanici nám stačí inicializovat framework se správnou konfigurací zařízení. Pro zjednodušení práce s používáním výsledné aplikace přidáme možnost přepnout do konfiguračního režimu a přepínání stavu zařízení. Možnost přepnutí do konfiguračního režimu umožníme pouze po spuštění zařízení. Pokud bude zmáčknuté tlačítko 1 vteřinu po spuštění programu, tak se přepneme do konfiguračního režimu. Pokud bychom chtěli zajistit větší bezpečnost, tak dáme přepnutí do konfiguračního režimu pouze pomocí Bluetooth mesh síti. To můžeme zařídit tak, že si přidáme vlastní stav zařízení, který přepne zařízení do konfiguračního režimu.

Přepínání stavu zařízení si namapujeme na stejné tlačítko. Tentokrát použijeme framework na detekci stisku tlačítka. Po stisknutí správného tlačítka si zjistíme v jakém stavu je naše zařízení, zvětšíme stav o 1 a nastavíme ho.

Barvu stanice si můžeme buď nastavit ve webové konfiguraci zařízení nebo až za běhu pomocí Bluetooth mesh (např. pomocí pomocné aplikace).

Program 2 Minimální program pro spuštění frameworku.

```
#include "interactive-mesh-framework.hpp"

using namespace imf;

// needs to be callable from C because of ESP-IDF framework
extern "C" void app_main(void)
{
    IMF imf {};
    // add devices
    imf.addDevice(
        DeviceType::Station,
        "XX:XX:XX:XX:XX:XX", 1,
        0x0002);
    // ...

    // register callbacks if wanted
    // imf.registerCallbacks(nullptr, nullptr,
    //      nullptr, nullptr, nullptr);

    imf.start();
}
```

4.2 Mobilní zařízení

Mobilní zařízení využívá větší část frameworku než stanice. Mobilní zařízení si uchovává informaci o nejbližším zařízení a podle toho změní svoji barvu.

Framework si inicializujeme stejně jako v projektu pro stanici. Navíc si při inicializaci zaregistrujeme callback funkci na události a update funkci. Z událostí nás zajímá pouze událost `DM_NEAREST_DEVICE_CHANGE`. Z události budeme ukládat nejbližší zařízení a čas události. V update callback funkci si budeme poté kontrolovat, jestli se nacházíme u správného zařízení. Abychom se vyhnuli šumu měření vzdálenosti, tak správné řešení budeme kontrolovat až po dostatečné době, co se nejbližší zařízení nezměnilo. V příkladové implementaci se osvědčilo nastavit tuto dobu na 5 vteřin.

Jakmile detekujeme, že je zařízení v blízkosti správné stanice, tak vybereme novou barvu. V příkladové implementaci se vybere nová barva náhodně ze seznamu barev stanic. V konečné podobě hry by se barva mohla vybírat i chytřeji, např. na základě polohy zařízení.

Výsledný program najdeme v příloze ve složce `src/finding-color/finding-color-mobile`.

5 Výsledky a diskuze

V této kapitole si navrhneme testování frameworku a následně budeme diskutovat výsledky testů.

Framework bude testován na příkladové aplikaci 'Najdi svoji barvu'. Pro testování bylo vyrobeno 10 zařízení, z toho 8 bylo použito jako stanice a 2 jako mobilní. Než započneme s testováním, musíme navrhnout rozmístění stanic, pořadí určení poloh zařízení a barvy stanic.

Rozmístění Rozmístění stanic má velký vliv na přesnost určení poloh. Z experimentálního měření vzdáleností (viz. obrázek 2.3) jsme zjistili, že pro nejlepší výsledky by měly být stanice rozmístěny alespoň s rozestupy 3 metry. Pro přesné měření vzdáleností by stanice měly mít na sebe přímou viditelnost. Případné překážky způsobí odrazy měřících signálů a tím bude měření vzdálenosti méně přesné. Každá stanice by tedy měla mít přímou viditelnost na alespoň 4 další stanice pro přesnější multilateraci.

Přesnost lokalizace výrazně záleží na prostředí, ve kterém jsou zařízení umístěna. Pokud se v prostředí šíří rádiový signál s odrazy, tak budou měření vzdálenosti nepřesné a tím pádem i lokalizace zařízení. Nejlepší prostředí je takové, kde není moc překážek, které rádiový signál můžou odrážet. Tedy ideální prostředí je například louka, kde zařízení umístíme na dřevěné tyče alespoň metr od země.

Postup určování polohy Jakmile jsou stanice rozmístěné, tak si stanice změří vzdálenosti k ostatním stanicím. Poté určíme v jakém pořadí si určí svoji polohu. První 3 stanice mají speciální význam. Určí nám počátek, směr osy x a směr osy y. Třetí zařízení umístíme na levou stranu osy x, aby pozice přímo odpovídaly realitě a nebyly zrcadlově převrácené (viz. kapitola 2.3). Každá další stanice musí mít v dosahu alespoň 3 další stanice s již určenou polohou. Pro nejlepší výsledky lokalizace mobilních zařízení ručně opravíme pozice stanic v pomocné aplikaci.

Barvy stanic Barvy stanic by měly být jednoduše rozlišitelné i v přímém slunci. To není jednoduché zaručit s použitými diodami. Na monitoru lze jednoduše rozlišit mezi žlutou (`#FFFF00`) a zelenou (`#00FF00`), ale na zařízeních barvy vypadá skoro stejně, zejména na přímém slunci. Z testů na zařízeních jsou snadno rozlišitelné následující barvy: `#FF0000`, `#00FF00`, `#0000FF`, `#FF00FF`, `#FFFFFF`.

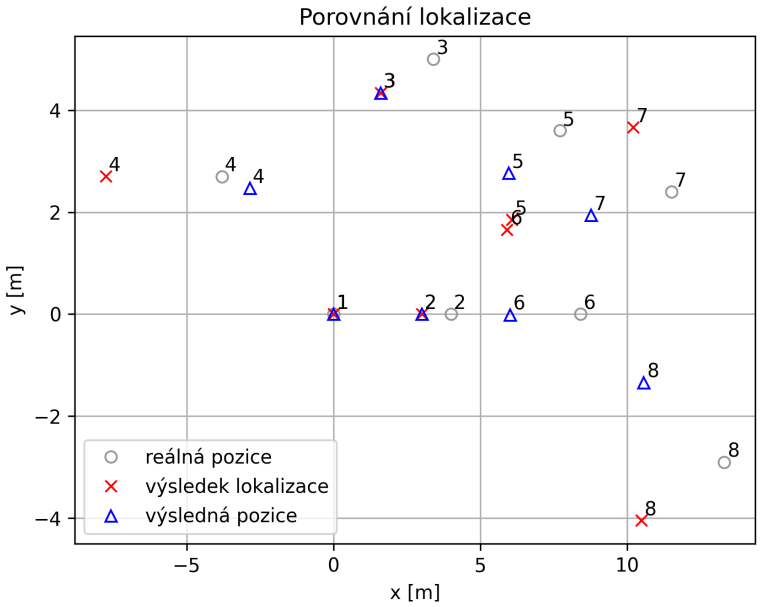
5.1 Výsledky

Lokalizace Výsledky lokalizace stanic dosahují přijatelných výsledků. Nepřesnosti měření vzdáleností se v lokalizaci projevily. Pozice stanic byly v průběhu postupně upraveny, aby více odpovídaly reálnému rozložení. Tím dosáhneme lepších výsledků lokalizace mobilních zařízení. Korekce pozic stanic nebyla provedena přesně podle reálných pozic, ale i tak se dosáhlo dobrých výsledků lokalizace mobilních zařízení. Srovnání reálných pozic stanic, výsledků lokalizace a ruční úpravy pozice můžeme vidět na obrázku 5.1. Ukázka následné cesty jednoho mobilního zařízení mezi několika stanicemi je na obrázku 5.2.

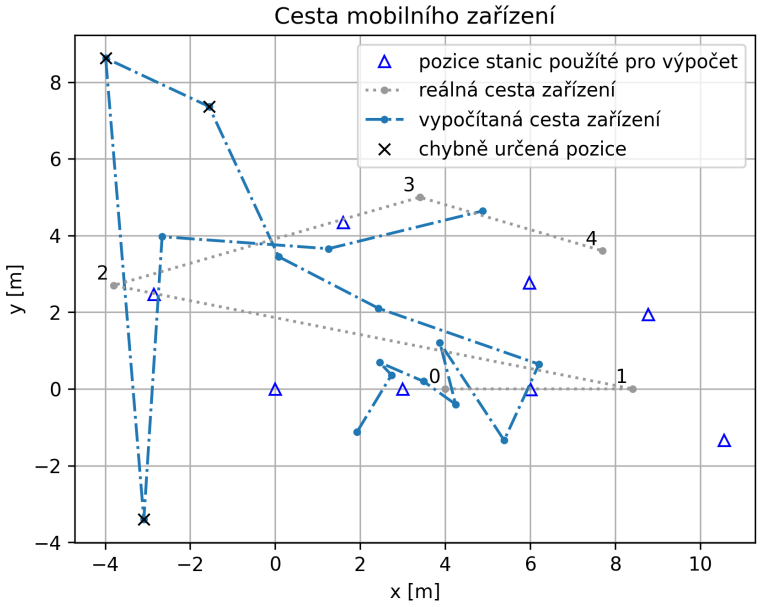
Některé chyby v pozicích jsou způsobeny zvoleným způsobem lokalizace. Pokud se kružnice jednotlivých bodů neprotínají v multilateraci, tak výsledné přímky lineárních rovnic neodpovídají hledání výsledného bodu. Můžeme to vidět na obrázcích 5.4 a 5.5. Dobré určení polohy můžeme vidět na obrázku 5.3. Výsledky multilaterace jsou srovnané s alternativní metodou pro hledání řešení soustavy rovnic multilaterace, konkrétně Levenberg-Marquardt (LM) algoritmem [20]. LM algoritmus nám určí poloho s menší chybou. Implementací tohoto algoritmu do frameworku lze přesnost lokalizace zlepšit. Hlavní nevýhodou algoritmu je jeho výkonová náročnost. Pro porovnání algoritmů byl LM algoritmus spuštěn na PC s daty ze zařízení.

Výkon aplikace Výkon aplikace budeme hodnotit hlavně na mobilních zařízení, protože v běžném provozu budou nejvíce zatížené. Stanice po určení své polohy pouze odpovídají na dotazy měření vzdálenosti. V mobilních zařízení se hlavní smyčka skládá z měření vzdáleností od všech stanic, určení polohy a provedení aplikační aktualizační funkce. Při testování aplikace se ukázalo, že celkově jedna smyčka trvá 2.8–3.3 vteřiny, což je pro naší aplikaci dostatečné. Nejvíce procesorového času program spotřebuje měřením vzdáleností ke stanicím, konkrétně zhruba 1.8 vteřiny. Určení polohy trvá dalších 0.5 vteřiny.

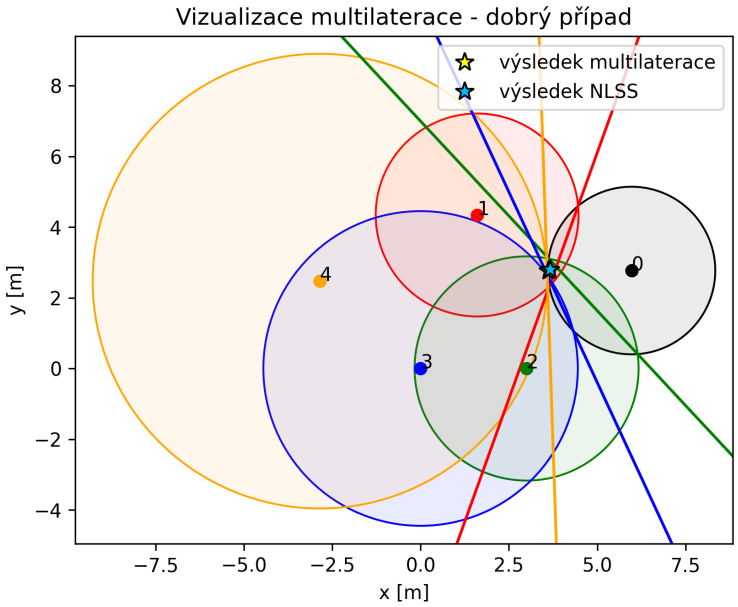
Měření vzdálenosti není možné více zrychlit se stávajícím způsobem měření. U měření vzdálenosti nastavujeme kolik měření se má provést a v jakém odstupu. Tohoto času bylo dosaženo nastavením nejmenšího možného počtu měření a nejmenším odstupem času. Jelikož měříme vzdálenosti ke každé stanici zvlášť, tak se čas přímo úměrně prodlužuje s počtem stanic v síti. Pokud by byly stanice řídce rozmístěné (s velkými rozestupy), tak můžeme ve frameworku zapnout měření vzdáleností pouze k dostupným stanicím. Skenování dostupných stanic je také časově náročné, tedy pro malý počet stanic se nevyplatí využívat této funkce. V tomto testu byl výkon dostatečný, takže tuto konfiguraci zařízení není potřeba nic měnit.



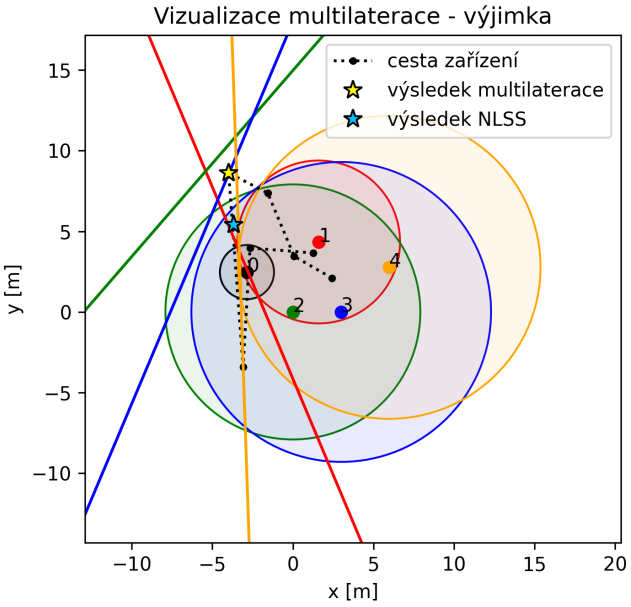
Obrázek 5.1 Porovnání pozic určených pomocí multilaterace, reálných pozic zařízení a pozic po korekci. Čísla u bodů znamenají pořadí určení poloh. Každá stanice určovala svoji polohu z finálních pozic předchozích stanic. Pozice po korekci nejsou stejné jako reálné pozice stanic. To je způsobené tím, že naměřené vzdálenosti zařízeními neodpovídají přesně reálné vzdálenosti. To aplikaci nevadí, protože sleduje pouze o lokální pozice.



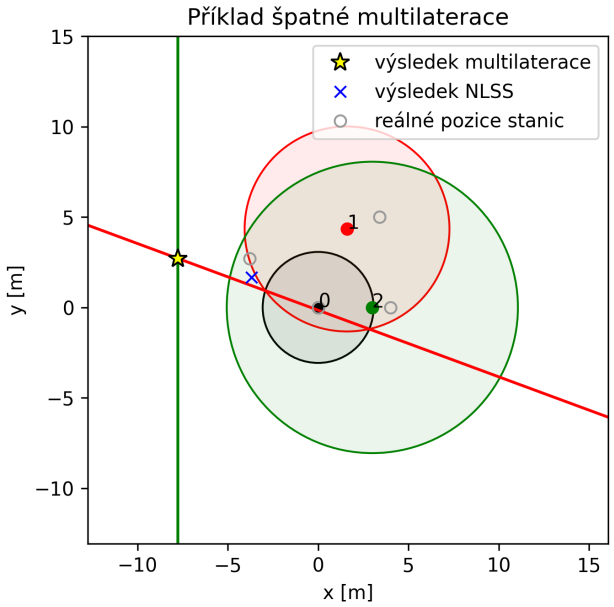
Obrázek 5.2 Obrázek ukazuje část cesty jednoho mobilního zařízení a cestu porovnává s odhadovanou cestou zařízení. Zařízení se pohybovalo mezi stanicemi podle barvy, kterou vygenerovalo zařízení. Body označené křížkem představují místa, kde došlo k největší chybě lokalizace (vznikly při stání u stanice označené číslem 2). Lokalizace jednoho takového bodu je vizualizována v obrázku 5.4. Animace celé cesty zařízení je v příloze ve složce `result`.



Obrázek 5.3 Obrázek ukazuje příklad správného určení pozice. Jednotlivé grafické prvky vizualizace jsou popsány u obrázku 1.6.



Obrázek 5.4 Obrázek ukazuje příklad špatně určené pozice. Můžeme vidět, že výsledná pozice je mimo průniky kružnic. To je způsobeno stanicí číslo 2, která se neprotíná s kružnicí stanice číslo 0. Přímka, která vznikne odečtením rovnic, vznikne mimo kružnice. Nepřesnost v měření může být způsobená buď šumem v měření, nebo tím, že se mobilní zařízení pohnulo mezi měřeními k jednotlivým stanicím.



Obrázek 5.5 Obrázek ukazuje příklad špatně určené pozice. Můžeme vidět, že výsledná pozice je mimo průniky kružnic. To je způsobeno stanicí čislo 2, která se neprotíná s kružnicí stanice číslo 0. Přímka, která vznikne odečtením rovnic, vznikne mimo kružnice.

Závěr

V této práci byl představen návrh frameworku pro síť světelných zařízení, který byl následně implementován a otestován. Pro komunikaci mezi zařízeními byla využita distribuovaná síť Bluetooth mesh. Dále pro měření vzdálenosti mezi zařízeními byla využita technologii WiFi FTM a pro lokalizační systém byla použita multilaterace. Zařízení bylo rozděleno na dva moduly, kde jeden zabezpečuje Bluetooth komunikaci a druhý běh frameworku a měření vzdálenosti. Pro komunikaci mezi moduly byla vyvinuta vlastní asynchronní sériová komunikace.

Pro framework byla navržena a vyrobena fyzická zařízení založená na ESP32-S3 modulech s RGB LED.

V rámci práce byla také vypracována pomocná aplikace na vizualizaci pozic jednotlivých zařízení s možností opravit jejich případné nepřesně určené umístění. Navíc do aplikace bylo přidáno ovládání vlastností dostupných z Bluetooth mesh sítě.

Implementace vytvořeného frameworku i navržená zařízení byla úspěšně otestována na ukázkové aplikaci „Najdi svoji barvu“. Pomocí aplikace jsme otestovali správnou funkčnost všech částí frameworku a také ověřili, že jeho výkon je pro interaktivní aplikaci dostatečný. Přesnost lokalizačního systému je ovlivněna velkým rozptylem měření vzdáleností mezi zařízeními. Pro přijatelnou přesnost určení poloh mobilních zařízení je proto zapotřebí při rozmisťování pevných stanic opravit jejich automaticky určené pozice. V této podobě je systém vhodný pro odhad pozic mobilních zařízení pohybujících se uvnitř plochy ohraničené pevnými stanicemi, kde je zajištěno dostatečné přesné měření vzdálenosti ke stanicím.

Budoucí práce

V této kapitole uvedeme některé možnosti budoucí práce či rozšíření, které by v dalším vývoji bylo vhodné vzít do úvahy. S ohledem na cíle této práce však jsou již nadstavbové a rozvíjející (a jejich implementace není pro použitelnost samotného frameworku nebo pilotní cílové aplikace nezbytná).

Zařízení Navržená zařízení byla vyrobena za přijatelné ceny i pro větší nasazení. Materiál na zvolená zařízení stál zhruba 700 Kč na jedno zařízení (i s baterií a dalším příslušenstvím). Nicméně pokud bychom dokázali cenu jednoho zařízení snížit, tak by se zvýšila možnost využití frameworku i při případných nárocích na nízký rozpočet. V budoucí práci můžeme prozkoumat možnosti sjednocení BLE-Mesh a WiFi na jedno zařízení nebo možnosti využití levnějších mikrokontrolérů. Pro Bluetooth Mesh může stačit levnější čip. Můžeme také prozkoumat možnosti změny struktury frameworku, např. přesunout Bluetooth mesh na hlavní modul a druhý modul by zabezpečoval pouze měření vzdálenosti.

Lokalizace Lokalizační systém vypracovaný v této práci je pro cílové využití dostatečný. V budoucí práci můžeme prozkoumat zlepšení lokalizace. Buď můžeme zlepšit stávající systém, např. filtrováním výsledné polohy (třeba pomocí Kalmanova filtru) nebo zlepšit měření vzdáleností. Existují přesnější technologie na měření vzdálenosti, např. UWB (Ultra-Wideband) technologie. U této technologie

je typicky měření vzdálenosti rychlejší a zároveň nabízí větší přesnost měření. Nevýhoda této metody je cena modulů implementující UWB. Například modul DWM3000 v menším množství (< 100 ks) stojí zhruba 600 Kč.

Správa zařízení Nasazení aplikací založených na našem frameworku by mohla být zjednodušená Android/iOS aplikací, která by zastupovala funkci pomocné aplikace. Navíc by automaticky nastavila nové zařízení a poskytovala by informace o stavu sítě.

Literatura

1. Espressif Systems. *ESP-IDF* [online]. 2024. [cit. 2024-01-07]. Dostupné z: https://github.com/espressif/esp-idf.
2. ASA, Nordic Semiconductor. *Wireless Connectivity Comparison* [online]. 2019. [cit. 2024-01-06]. Dostupné z: https://web.archive.org/web/20240106101329/https://f.hubspotusercontent10.net/hubfs/1961165/Tabell%20Wireless%20Connectivity%20Comparison%20190426.pdf.
3. Espressif Systems. *ESP-WIFI-MESH* [online]. 2023. [cit. 2024-01-06]. Dostupné z: https://web.archive.org/web/20240106095134/https://docs.espressif.com/projects/esp-idf/en/v5.1.2/esp32s3/api-guides/esp-wifi-mesh.html.
4. Espressif Systems. *ESP-BLE-MESH* [online]. 2023. [cit. 2024-01-09]. Dostupné z: https://web.archive.org/web/20240109172351/https://docs.espressif.com/projects/esp-idf/en/v5.1.2/esp32s3/api-guides/esp-ble-mesh/ble-mesh-index.html.
5. Bluetooth SIG. *Bluetooth Mesh Networking FAQs* [online]. 2024. [cit. 2024-01-10]. Dostupné z: https://web.archive.org/web/20240110090454/https://www.bluetooth.com/learn-about-bluetooth/feature-enhancements/mesh/mesh-faq/.
6. *Thread official website* [online]. [cit. 2024-05-03]. Dostupné z: https://www.threadgroup.org/.
7. *Zigbee official website* [online]. [cit. 2024-05-03]. Dostupné z: https://csa-iot.org/all-solutions/zigbee/.
8. Woolley, Martin. *Bluetooth Mesh Models - Technical Overview* [online]. [cit. 2024-05-06]. Dostupné z: https://web.archive.org/web/20240506155249/https://www.bluetooth.com/wp-content/uploads/2019/04/1903_Mesh-Models-Overview_FINAL.pdf.
9. Afaneh, Mohammad. *Bluetooth Mesh Networking: The Ultimate Guide* [online]. 2022. [cit. 2024-01-05]. Dostupné z: https://web.archive.org/web/20240105092233/https://novelbits.io/bluetooth-mesh-networking-the-ultimate-guide/.
10. Kei Ren, Martin Woolley. *The Fundamental Concepts of Bluetooth Mesh Networking Part 1* [online]. 2017. [cit. 2024-01-05]. Dostupné z: https://web.archive.org/web/20240105172921/https://www.bluetooth.com/blog/the-fundamental-concepts-of-bluetooth-mesh-networking-part-1/.
11. Kei Ren, Martin Woolley. *The Fundamental Concepts of Bluetooth Mesh Networking Part 1* [online]. 2017. [cit. 2024-01-05]. Dostupné z: https://web.archive.org/web/20240105172941/https://www.bluetooth.com/blog/the-fundamental-concepts-of-bluetooth-mesh-networking-part-2/.

12. KASTNES, Pål. *Using the Nordic Distance Toolbox to measure the distance between short-range radios* [online]. 2023. [cit. 2024-01-12]. Dostupné z: https://web.archive.org/web/20240112093406/https://blog.nordicsemi.com/getconnected/using-the-nordic-distance-toolbox-to-measure-the-distance-between-short-range-radios.

13. LI, Guoquan; GENG, Enxu; YE, Zhouyang; XU, Yongjun; LIN, Jinzhao; PANG, Yu. Indoor Positioning Algorithm Based on the Improved RSSI Distance Model. *Sensors*. 2018, roč. 18, č. 9, s. 2820. ISSN 1424-8220. Dostupné z DOI: 10.3390/s18092820.

14. XIA, Shixiong; LIU, Yi; YUAN, Guan; ZHU, Mingjun; WANG, Zhaohui. Indoor Fingerprint Positioning Based on Wi-Fi: An Overview. *ISPRS International Journal of Geo-Information*. 2017, roč. 6, č. 5, s. 135. ISSN 2220-9964. Dostupné z DOI: 10.3390/ijgi6050135.

15. SANG, Cung Lian; ADAMS, Michael; HESSE, Marc; HORMANN, Timm; KORTHALS, Timo; RUCKERT, Ulrich. A Comparative Study of UWB-based True-Range Positioning Algorithms using Experimental Data. In: *2019 16th Workshop on Positioning, Navigation and Communications (WPNC)*. IEEE, 2019. Dostupné z DOI: 10.1109/wpnc47567.2019.8970249.

16. ESPRESSIF SYSTEMS. *RF Coexistence* [online]. 2023. [cit. 2024-01-07]. Dostupné z: https://web.archive.org/web/20240107133907/https://docs.espressif.com/projects/esp-idf/en/v5.1.2/esp32s3/api-guides/coexist.html.

17. BOURKE, Paul. *Circles and spheres* [online]. [cit. 2024-05-05]. Dostupné z: https://paulbourke.net/geometry/circlesphere/.

18. ESPRESSIF SYSTEMS. *Logging library* [online]. 2023. [cit. 2024-01-11]. Dostupné z: https://web.archive.org/web/20240111152110/https://docs.espressif.com/projects/esp-idf/en/v5.1.2/esp32s3/api-reference/system/log.html.

19. ESPRESSIF SYSTEMS. *Non-volatile Storage Library* [online]. [cit. 2024-05-07]. Dostupné z: https://docs.espressif.com/projects/esp-idf/en/v5.1.3/esp32/api-reference/storage/nvs_flash.html.

20. MORÉ, Jorge J. The Levenberg-Marquardt algorithm: Implementation and theory. In: WATSON, G. A. (ed.). *Numerical Analysis*. Berlin, Heidelberg: Springer Berlin Heidelberg, 1978, s. 105–116. ISBN 978-3-540-35972-2. Dostupné z DOI: 10.1007/BFb0067700.

21. ESPRESSIF SYSTEMS. *ESP-IDF Get Started* [online]. 2024. [cit. 2024-05-04]. Dostupné z: https://docs.espressif.com/projects/esp-idf/en/v5.1.3/esp32s3/get-started/index.html.

22. *nRF Mesh aplikace* [online]. [cit. 2024-05-05]. Dostupné z: https://www.nordicsemi.com/Products/Development-tools/nRF-Mesh.

A Přílohy

Příloha elektronické verze obsahuje následují složky:

- `boards` - Zdrojové soubory zařízení popsaných v kapitole 3.
- `distance-analysis` - Data z experimentálního měření vzdálenosti pomocí WiFi FTM a program, který data zpracovává. Výsledky analýzy dat jsou zmíněné v textu.
- `mesh-visualizer` - Zdrojové kódy pomocné aplikace, která byla v textu diskutována.
- `result` - Logy ze zařízení, které vznikly při testování frameworku, společně s programem na zpracování logů. Dále složka obsahuje animaci vypočítané cesty jednoho z mobilních zařízení.
- `src` - Zdrojové kódy frameworku a příkladových aplikací.
  - `src/ble-mesh` - Zdrojové kódy pro vedlejší modul a abstrakci HSL modelu Bluetooth mesh.
  - `src/common-components` - Zdrojové kódy implementace frameworku.
  - `src/example-project` - Příkladový projekt využívající náš framework, který můžeme využít pro vytváření nových projektů.
  - `src/finding-color` - Příkladové aplikace „Najdi svoji barvu“, na kterých jsme v textu testovali framework.
  - `src/scripts` - Pomocné shell skripty pro nahrání kódu na více připojených zařízení zároveň.

A.1 Kompilace a nahrání programu na zařízeni

Na kompilaci programu potřebujeme nainstalovat framework ESP-IDF od Espressif podle instrukcí dokumentace ESP-IDF [21]. Tato práce byla vyvíjena a testována na verzi v5.1.3.

Pro kompilaci a nahrání programů na zařízení postupujte podle následujícího návodu. Návod je tvořený pro Linux, pro ostatní systémy konzultujte dokumentaci výrobce zařízení [21].

```
source $HOME/esp/esp-idf/export.sh
idf.py build flash monitor
```

K `idf.py` přidejte `-p <PORT>` pro vybrání konkrétního připojeného zařízení, `<PORT>` nahraďte sériovým portem příslušící k zařízení (např. `/dev/ttyACM0` na Linuxu).

Ukázkové programy jsou ve složkách `src/finding-color/finding-color-station` a `src/finding-color/finding-color-mobile`.

A.2 Vytvoření nového projektu

Pro vytvoření nového projektu s naším frameworkem, zkopírujeme si příkladový projekt `src/example_project`.

1. Zkopírujeme si příkladový projekt `src/example_project`.
2. V našem projektu změníme název projektu v souboru `CMakeLists.txt`. Změníme název z `example-project` na řádku `project(example-project)` na název naší aplikace.
3. Pokud je projekt v jiné složce než příkladový projekt, tak změníme cestu k frameworku v souboru `CMakeLists.txt`. Změníme cestu na řádku začínajícím `list(APPEND EXTRA_COMPONENT_DIRS`. Cesta musí vést na složku `src/common-components` zdrojových souborů frameworku.
4. V souboru `sdkconfig.defualts` nastavíme typ zařízení, pro který vytváříme tento projekt. Projekt je v základu nakonfigurovaný pro stanici, pro změnu na mobilní zařízení nahradíme `CONFIG_IMF_STATION_DEVICE` za `CONFIG_IMF_MOBILE_DEVICE`.

A.3 Uživatelská dokumentace ukázkové aplikace

Zařízení obsahuje dvě LED, jednu na každém modulu. LED na hlavním modulu nám zobrazuje stav aplikace podle tabulky 2.2 nebo vlastních stavů. LED na Bluetooth modulu nám zobrazuje barvu zařízení.

A.3.1 Konfigurační režim

Zařízení můžeme přepnout do konfiguračního režimu pomocí zmáčknutí pravého tlačítka na hlavním modulu a po půl vteřině zmáčknutím a držením levého tlačítka. Jakmile se na zařízení LED rozsvítí světle růžově, tak tlačítko můžeme pustit. Pokud si zařízení vytvořilo vlastní WiFi, tak po připojení k zařízení najdeme webovou konfiguraci na adrese `192.168.4.1`. V ostatních případech si adresu zařízení můžeme zjistit v sériové komunikace zařízení.

Po otevření webové konfigurace, se postupně načtou jednotlivá nastavení. Po načtení můžeme hodnoty jednotlivých nastavení měnit. Pro uložení hodnot klikneme na tlačítko *Save*.

Ve webovém rozhraní si můžeme zobrazit i uložené logy kliknutím na odkaz *Show logs*. Pokud chceme logy vymazat, klikneme na tlačítko *Delete logs*.

Pro vystoupení z konfiguračního režimu zařízení restartujeme.

A.3.2 Příprava zařízení

Po výrobě zařízení se musí zařízení správně nastavit.

Nahrajeme aplikaci `src/ble-mesh/mesh-firmware/` na všechny vedlejší moduly. Pokud propojení modulů není podle tabulky 3.1, tak nastavíme pomocí příkazu `idf.py menuconfig` správnou konfiguraci pinů pro komunikaci, konkrétně nastavení `SERIAL_TX_GPIO` a `SERIAL_RX_GPIO`.

Nahrajeme mobilní verzi aplikace a aplikaci pro stanice na hlavní moduly zařízení (na každý modul pouze jednu verzi podle typu zařízení). Pokud připojení vedlejšího modulu není podle tabulky 3.1, tak nastavíme novou konfiguraci pinů pro komunikaci pomocí příkazu `idf.py menuconfig`, konkrétně nastavení `IMF_SERIAL_TX_GPIO` a `IMF_SERIAL_RX_GPIO`.

Vytvoříme si Bluetooth mesh síť a zapojíme do ní každý Bluetooth modul. Tento krok provedeme pouze po prvním spuštění zařízení. Poté se údaje na připojení k síti zachovají v paměti i po nahrání nové verze aplikace na vedlejší modul. Pro nastavení Bluetooth mesh využijeme mobilní aplikaci na Android i iOS *nRF Mesh* od Nordic Semiconductor [22]. Každé zařízení připojíme do naší sítě pomocí následujícího návodu[1]:

1. Klikneme na tlačítko *Add node* v záložce *Network*.
2. Vybereme si ze seznamu zařízení čekající na připojení k síti naše zařízení, které mají název *IMF Node*.
3. Po vybrání zařízení se zobrazí obrazovka s podrobnostmi o vybraném zařízení. Klikneme na tlačítko *Identify*, po kliknutí začne na zařízení blikat LED.
4. Na obrazovce si najdeme hodnotu *Unicast address* a napíšeme si ji nejlépe fyzicky na zařízení, později to budeme potřebovat na spárování modulů.
5. Klikneme na tlačítko *Provision*, tím se zařízení přidá do naší mesh sítě.
6. Dále potřebujeme nakonfigurovat jednotlivé modely na zařízení. V seznamu zařízení v záložce *Network* klikneme na nově přidané zařízení.
7. V části *Elements* si rozbalíme všechny elementy a postupně klikneme na každý model.
8. Po kliknutí na model se zobrazí podrobnosti modelu. Klikneme na tlačítko *Bind Key* a vybereme náš klíč *App key* (ve výchozím stavu bude v seznamu pouze jeden).

Jakmile máme nastavené Bluetooth moduly, tak si vytvoříme seznam párování Bluetooth modulů na hlavních modulů. Pro každé zařízení si zaznamenáme MAC adresu WiFi AP hlavního modulu a adresu Bluetooth modulu. Obě adresy můžeme zjistit přepnutím zařízení do konfiguračního režimu (viz. příloha A.3.1). Vytvoří se WiFi AP s názvem `NODE-XXXXXXXXXXXX`, kde `XX` jsou části MAC adresy v obráceném pořadí bytů. Heslo na Wifi najdeme v konfiguraci projektu aplikace pomocí `idf.py menuconfig` v položce `WIFI_CONNECT_AP_DEFAULT_PASSWORD`. Adresu Bluetooth modulu máme buď poznačenou z inicializace Bluetooth mesh, nebo si ji můžeme zjistit z webové konfigurace. Po připojení na WiFi konkrétního zařízení najdeme konfigurační stránku na adrese `192.168.4.1`. Zde můžeme najít adresu Bluetooth modulu připojeného k tomuto zařízení v položce `ble_mesh/addr`. Pokud adresa v políčku není, tak to znamená, že buď aplikace nebyla nikdy spuštěná v normálním režimu nebo je problém s komunikací s Bluetooth modulem.

Jakmile máme poznamenané párování adres jednotlivých modulů, tak si upravíme kód aplikací s naší konfigurací zařízení. V projektu pro mobilní zařízení i

[1]Návod je tvořený pro Android verzi aplikace, v iOS verzi se aplikace může mírně lišit.

stanici upravíme soubor `main/main.cpp`. Ve funkci s názvem `main` upravíme řádky `s_imf->addDevice` tak, aby zde byli informace o našich modulech — správná MAC adresa, kanál WiFi a Bluetooth mesh adresa. Aktualizovanou aplikaci nahrajeme do všech zařízení. Framework bude pracovat pouze se zařízeními, které jsou přidána při inicializaci, takže pokud tam nějaké zařízení nebude, tak s ním aplikace nebude komunikovat.

Jako poslední krok přípravy zařízení můžeme ve webové konfiguraci nastavit každé stanici její základní barvu, která se nastaví po spuštění zařízení. Tím si usnadníme práci při rozmisťování zařízení po herní ploše.

A.3.3 Provoz aplikace

Pro provozování aplikace budeme postupovat podle následujícího návodu.

1. Rozmístíme stanice a zapneme je.
2. Zkontrolujeme barvy stanic. Pokud jsou stanice s stejnými barvami blízko sebe, tak je přemístíme.
3. Přepneme všechny stanice do stavu měření vzdáleností. Buď pomocí levého tlačítka na hlavním modulu zařízení nebo prostřednictvím pomocné aplikace v PC.
4. Počkáme 3 minuty pro dostatečný počet měření.
5. Postupně budeme přepínat stanice do stavu určení polohy (stav č. 2). Jakmile je poloha určená, přepneme dané zařízení do stavu č. 3, tj. běh výsledné aplikace.
6. Po každém určení polohy stanice zkontrolujeme polohu zařízení v pomocné aplikace a případně ji opravíme.
7. Jakmile jsou určené polohy všech stanic, zapneme mobilní zařízení.
8. Spustíme hru přepnutím mobilních zařízení do třetího stavu (běh aplikace).