Univerzita Hradec Králové
Fakulta informatiky a managementu
Katedra informatiky a kvantitativních metod

Vývoj 3D trackeru kompatibilního s HTC Vive

Diplomová práce

Autor: Bc. Martin Donát
Studijní obor: Aplikovaná informatika

Vedoucí práce: Ing. Pavel Kříž, Ph.D.

Hradec Králové duben 2020

Prohlášení:

Prohlašuji, že jsem diplomovou práci zpracoval samostatně a s použitím uvedené literatury.

V Hradci Králové dne 30.4.2020 Martin Donát

Poděkování:

Rád bych poděkoval vedoucímu diplomové práce Ing. Pavlu Křížovi, Ph.D. za velice cenné rady, které mi pomohly úspěšně dokončit tuto práci a za trpělivost při dlouhých konzultacích. Dále bych rád poděkoval své rodině, která mě podporovala a bez které bych tuto práci nedokončil.

Děkuji též spolku hkfree.org za zapůjčení osciloskopu Rigol, který byl využit k oživení vyvíjeného trackeru.

Anotace

Cílem diplomové práce bylo s použitím vhodného Bluetooth Low Energy čipu navrhnout a implementovat vlastní 3D tracker kompatibilní s HTC Vive. Diplomová práce se nejprve věnuje seznámení s technologií Bluetooth Low Energy a 3D trackování pomocí HTC Vive. Dále následuje analýza a návrh řešení vlastního 3D trackeru, včetně návrhu mobilní aplikace pro operační systém Android. Poté je v práci uveden podrobný popis implementace včetně ukázek zdrojových kódů. Vytvořený 3D tracker je schopný trackovat vlastní pozici v reálném čase s přesností na jeden milimetr. 3D tracker byl vytvořen jako wearable zařízení a je tak možné jej nosit na ruce podobně jako hodinky. Naprogramovaná mobilní aplikace umožňuje práci s geometrií Base Station, která je potřebná pro samotné trackování pozice. Mobilní aplikace dále slouží pro vykreslování aktuální pozice 3D trackerů v prostoru.

Annotation

Title: Development of 3D tracker compatible with HTC Vive

The aim of the diploma thesis was to design and implement own 3D tracker compatible with HTC Vive using a suitable Bluetooth Low Energy chip. The diploma thesis deals first with the introduction of Bluetooth Low Energy technology and 3D tracking using HTC Vive. Then follows second part with the analysis and design of own 3D tracker, including the design of a mobile application for the Android operating system. The work contains detailed description of the implementation, including examples of source codes. The created 3D tracker is able to track its own position in real time with an accuracy of one millimeter. The 3D tracker was created as a wearable device and thus can be worn on your wrist like a watch. The programmed mobile application allows you to work with the geometry of the Base Station, which is needed for the position tracking itself. The mobile application is also used to draw the current position of 3D trackers in space.

Obsah

Seznam obrázků

Seznam tabulek

1 Úvod

V dnešní době se virtuální realita pomalu stává poměrně rozšířenou technologií. Tato technologie uživateli umožňuje vnímat prostředí a pracovat s ním tak, jak by s ním interagoval v reálném světě. Virtuální realitu je možné využívat například pro pořádání online konferencí, pro spolupráci při modelování budov, pro online komunikaci, pro hraní her a mnoho dalšího. V současnosti je set pro virtuální realitu HTC Vive bohužel velice nákladnou záležitostí, nicméně snad do budoucna budou ceny těchto headsetů pro virtuální realitu postupně klesat.

Diplomová práce se bude zabývat návrhem a implementací vlastního 3D trackeru s použitím vhodného Bluetooth Low Energy čipu. Navrhovaný 3D tracker bude kompatibilní s HTC Vive, pro trackování vlastní pozice bude využívat zařízení Base Station vyráběné pro VR set HTC Vive. Tento tracker by měl být co nejmenší tak, aby bylo možné využívat ho například jako náramek na ruku. 3D tracker by měl být schopen pomocí Bluetooth Low Energy komunikovat se smartphonem. Na základě toho bude vytvořena také mobilní aplikace pro operační systém Android. Aplikace bude umožňovat komunikaci s 3D trackerem a dále bude sloužit především pro zobrazování pozice 3D trackeru v prostoru.

2 Cíl práce

Cílem diplomové práce je s použitím vhodného Bluetooth Low Energy čipu navrhnout a implementovat vlastní 3D tracker kompatibilní s HTC Vive. Navrhovaný 3D tracker bude využívat pro trackování pozice zařízení Base Station vyráběné společností HTC pro VR set HTC Vive. Součástí práce bude také vědecko-popularizační poster, který bude prezentovat základní principy fungování a celé řešení. V diplomové práci bude zhotovena také mobilní aplikace, která bude sloužit ke komunikaci s 3D trackerem. Mobilní aplikace bude dále sloužit pro zobrazování pozice 3D trackeru v prostoru. 3D tracker bude komunikovat s chytrým telefonem pomocí Bluetooth Low Energy.

3 Technologie Bluetooth Low Energy (BLE) na platformě chipů Nordic Semiconductor

3.1 Bluetooth Low Energy

První verze technologie Bluetooth byla vydána roku 1999. Od tohoto roku vzniklo několik dalších verzí technologie Bluetooth. V posledních verzích technologie Bluetooth se rozlišuje hlavně mezi dvěma standardy. Jedná se o Bluetooth Classic a Bluetooth Low Energy, dále jen BLE. Nejnovější verze technologie Bluetooth je verze 5, která podporuje tzv. Bluetooth mesh sítě umožňující nasazení rozsáhlých sítí mezi zařízeními [1].

Jedná se o technologii pro bezdrátový přenos dat mezi zařízeními na krátkou vzdálenost. Bluetooth je spravován konsorciem SIG (Special Interest Group), které dnes sdružuje desítky tisíc společností, viz [2]. Technologie Bluetooth je standardizována jako IEEE 802.15.1, přičemž využívá bezlicenčního pásma 2,4 GHz rozděleného do 79 kanálů s rozestupem 1 MHz. Protože toto pásmo využívají i další technologie pro bezdrátový přenos, je využíváno pro komunikaci všech 79 kanálů, které se během komunikace několikrát změní. Jedná se o techniku FHSS (Frequency Hopping Spread Spectrum). Následující kanál pro vysílání je určen pseudonáhodným číselným generátorem na zařízení typu master. Ostatní zařízení, která jsou připojena k master zařízení jsou zařízení typu slave [3].

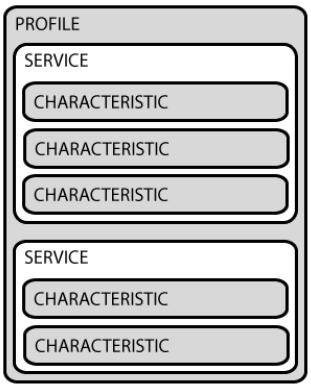
Standard BLE byl navržen pro energeticky nenáročné aplikace. Aby BLE zajistilo spolehlivý přenos v pásmu 2,4 GHz, využívá robustní techniky FHSS. Oproti Bluetooth Classic využívá BLE pro přenos dat pouze 40 kanálů s rozestupem 2 MHz v rámci 2,4 GHz pásma. Standart BLE poskytuje vývojářům velké množství flexibility zahrnující několika vrstvou PHY (Physical layer), která podporuje přenos dat rychlostí od 125 Kb/s do 2 Mb/s. Dále vrstva PHY umožňuje vysílat s různou úrovní vysílacího výkonu od 1 mW do 100 mW. V neposlední řadě se vrstva PHY stará o zabezpečení. BLE podporuje několik topologií sítě, mezi které patří point-to-point, broadcast a mesh [4].

V Bluetooth Low Energy může být zařízení v jedné ze čtyř rolí. Jedná se o role peripheral, central, broadcaster a observer. Zařízení v roli peripheral běžně

poskytuje služby a data ostatním bluetooth zařízením skrze služby a charakteristiky. Tato zařízení neiniciují připojení k ostatním bluetooth zařízením, ale naopak čekají jako server, než k nim dorazí požadavek o připojení. Peripheral zařízení můžou jednou za čas dát ostatním zařízením v okolí vědět, že existují a mají data a služby pro ostatní zařízení. Zařízení, která se připojují k peripheral zařízením, jsou zařízení v roli central. Tato zařízení mohou skenovat blízká okolní bluetooth zařízení a iniciovat spojení s nimi v případě, že je to možné. Central zařízení se chovají jako klienti, kteří vysílají požadavky na peripheral zařízení skrze služby a charakteristiky. Další rolí, ve které se může nacházet zařízení, je role broadcaster. Tato role umožňuje zařízení pouze sdílet data s okolními zařízeními skrze posílané advertise pakety[1]. Takto sdílená data jsou zachytitelná všemi ostatními zařízeními. Broadcaster zařízení nemůže být spojeno s žádným okolním zařízením. Poslední rolí je role observer. Jedná se o opak role broadcaster. V této roli zařízení jen pasivně poslouchá advertise pakety, ze kterých zpracovává přibalená data. V této roli také není možné, aby bylo navázáno spojení s ostatními zařízeními [3, 5].

Aby mohla peripheral zařízení poskytovat data, jsou k tomu nutné služby a charakteristiky. Na nejnižší úrovni se nacházejí charakteristiky. Charakteristiky umožňují čtení a zapisování dat. Každá charakteristika má vlastní jedinečný identifikátor, kterým je UUID (Universally Unique Identifier). UUID je 16 bitové pro standartní služby/charakteristiky stanovené organizací SIG nebo 128 bitové pro vlastní služby/charakteristiky. Vlastní služby a charakteristiky mohou být libovolně definovány. Každá charakteristika musí být definována v rámci určité služby. Do jedné služby je možné zahrnout vícero souvisejících charakteristik. Pokud je více souvisejících služeb, je možné je seskupit do jednoho profilu. Celá hierarchie profilu, služeb a charakteristik je znázorněna na obrázku 1 [3].

[1] Pakety, které vysílá zařízení do okolí, aby se ostatní zařízení o vysílajícím zařízení dozvěděla.



Obrázek 1 Znázornění hierarchie profilu, služeb a charakteristik. Zdroj [3]

3.2 Platforma Nordic Semiconductor

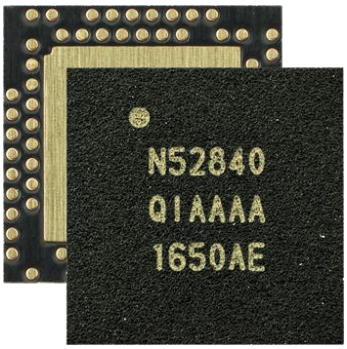
Jelikož se Bluetooth skládá ze dvou celků – hardwaru a softwaru, není tomu jinak ani na platformě od firmy Nordic Semiconductor. Nordic Semiconductor vyrábí celou řadu mikrokontrolerů podporujících různé bezdrátové standardy. Jedná se například o standardy Bluetooth, Bluetooth Low Energy, Zigbee a NFC. Aby tyto mikrokontrolery mohly fungovat, je nezbytná přítomnost druhého celku – softwaru. Nordic Semiconductor proto vyvíjí vlastní SDK (Software Development Kit), které se skládá z bohatého vývojářského prostředí, ovladačů pro desky, knihoven, příkladů zdrojových kódů pro práci s periferiemi, SoftDevices a proprietárních rádiových protokolů. Celé SDK je dostupné ke stažení jako ZIP archiv. Díky tomu je možné, aby si vývojář mohl svobodně zvolit IDE (Integrated Development Environment) a kompilátor dle vlastního výběru [6].

Stěžejní část SDK tvoří SoftDevice. Jedná se o předkompilovaný binární software, který implementuje výše zmíněné bezdrátové protokoly. SoftDevice taktéž snižuje čas vývojáři potřebný pro kompilaci projektu. Dále SoftDevice poskytuje run-time izolaci od hardwaru, na kterém poté výsledná aplikace běží. Aplikaci poskytuje SoftDevice API (Application Programming Interface) jako rozhraní vyššího programovacího jazyka například jako hlavičkový soubor jazyka C [7].

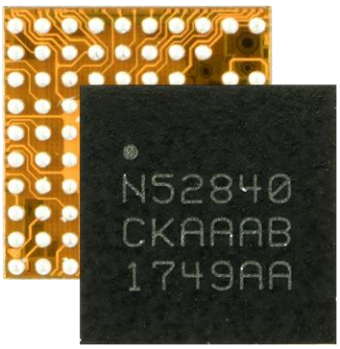
3.2.1 nRF52840

Jedná se o nejpokročilejší mikrokontroler, dále jen MCU, z rodiny čipů nRF52 SoC (System On Chip). nRF52840 je více protokolový MCU s plným paralelním

během protokolů. Tento MCU podporuje bezdrátové protokoly Bluetooth 5, Bluetooth mesh, Thread, Zigbee, 802.15.4, ANT a 2,4 GHz proprietární stack. Vysílací výkon MCU může být až +8 dBm. Samotný MCU je postavený na 32 bitovém ARM Cortex-M4 procesoru s plovoucí řádovou čárkou běžícím na taktu 64 MHz. Dále je v nRF52840 k dispozici až 1 MB flash paměti a až 256 KB RAM (Random Access Memory) paměti pro náročné aplikace. Mezi další přednosti tohoto MCU lze zařadit NFC-A Tag pro jednoduché párování či platební aplikace, kryptografickou jednotku umožňující širokou škálu kryptografických operací, které jsou vykonávány velmi efektivně nezávisle na procesoru a velké množství periferií. Jedná se např. o sběrnici I2S, TWI (I2C), SPI, UART, QSPI, PWM, PDM, 12 bitové ADC či USB 2.0. O nízkou spotřebu MCU se stará sofistikovaný power management. nRF52840 je dostupný ve dvou pouzdrech – QFN73 a WLCSP. Tato pouzdra je možné vidět na obrázcích 2 a 3 [8].



Obrázek 2 Provedení nRF52840 v pouzdru QFN73 (7x7 mm). Zdroj [8]



Obrázek 3 Provedení nRF52840 v pouzdru WLCSP (3,5x3,6 mm). Zdroj [8]

4 3D tracking pomocí zařízení HTC Vive

Protože hlavním cílem práce je vytvořit 3D tracker kompatibilní s HTC Vive, bude v této kapitole představena technologie pro 3D tracking, která bude použita pro vytvoření vlastního 3D trackeru. Zařízení HTC Vive využívají technologii pro 3D tracking nazývanou jako Lighthouse tracking či SteamVR tracking. Jedná se o technologii vyvinutou společností Valve. V současné době existují dvě verze technologie SteamVR tracking. Protože druhá verze vylepšuje verzi první, bude nejprve popsána první verze. Následně budou uvedeny rozdíly druhé verze od verze první. Poté bude představen detailní princip fungovávání Lighthouse stanic. Software využívaný pro trackování, nazývaný jako SteamVR, je proprietární a není veřejně dostupný. Avšak open source komunita provedla reverse engineering protokolu, kterým spolu komunikují Lighthouse stanice a senzory. Práce se tak bude opírat o neoficiální informace poskytované open source komunitou. Kapitoly zabývající se SteamVR trackingem byly zpracovány podle [9].

4.1 SteamVR tracking 1.0

V této technologii je 3D tracking založený na dvou druzích agentů. Prvním typem agenta jsou tzv. Lighthouse stanice. Každá Lighthouse stanice obsahuje několik infračervených (IR) LED diod a dva malé motory vrhající laserové paprsky do místnosti ve dvou směrech – horizontálně a vertikálně. Lighthouse stanice ozáří místnost následujícím způsobem. Nejprve se rozsvítí IR LED diody, poté je vržen laserový paprsek do místnosti v jednom směru, následuje opět rozsvícení IR LED diod a poté je do místnosti vržen laserový paprsek ve směru druhém. Tento cyklus ozařování místnosti se poté stále dokola opakuje. Druhým typem agentů jsou různé senzory na headsetu[2] a VR ovladačích. V první verzi SteamVR trackingu se využívají senzory TS3633 vyráběné firmou Triad Semiconductor.

Celý princip 3D trackingu je založený na interakci výše zmíněných agentů. Nejprve Lighthouse rozsvítí IR LED diody, tím je vyvolán reset na všech senzorech, které se v dané oblasti nachází. To způsobí, že se senzory nejprve vynulují a poté

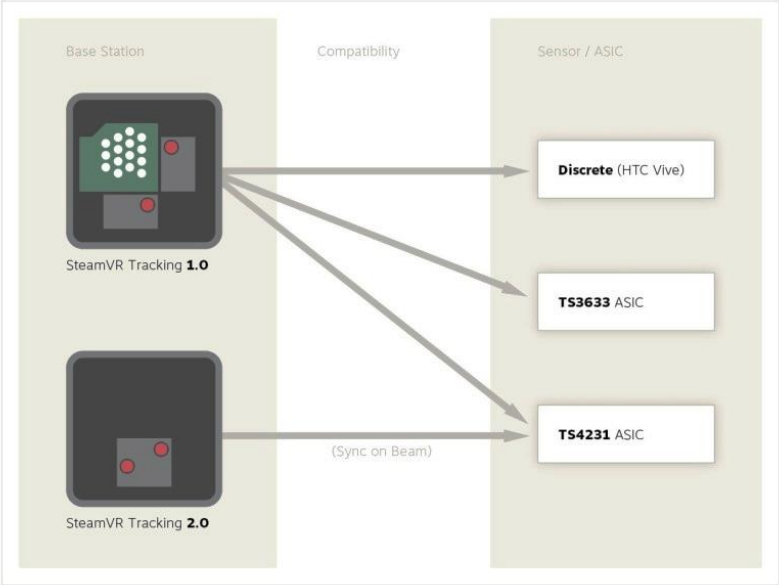
[2] Zařízení obsahující displeje, sluchátka a další senzory, které si uživatel nasadí na hlavu. Díky tomu se uživatel ocitne ve virtuální realitě.

začnou počítat. Senzory počítají tak dlouho, dokud nejsou zasaženy hýbajícím se laserovým paprskem, který je vrhán do místnosti Lighthousem nejprve v horizontálním směru. V momentně, kdy jsou senzory zasaženy laserovým paprskem je možné určit úhel mezi Lighthousem a senzorem, neboť laserový paprsek je vrhán do místnosti při konstantní rychlosti otáček motoru, aby pokryl úhel 180°. Následuje resetující pulz rozsvícením IR LED diod Lighthousem. Senzory jsou opět vyresetovány a začínají znovu počítat. Následuje opět vrhání laserového paprsku do místnosti, ale tentokrát ve vertikálním směru. I v tomto směru je laserový paprsek vrhán do místnosti při konstantní rychlosti a tak, aby pokryl úhel 180°. Jakmile jsou senzory zasaženy laserovým paprskem, přestanou počítat a je možné určit úhel ve vertikálním směru mezi Lighthousem a senzory. Poté se celý cyklus opakuje. Takto získaná data jsou poté předána ze senzorů do centrální jednotky, která je poté schopna pomocí matematických výpočtů, případně i z dat z IMU (Inertial Measurement Unit) senzorů rekonstruovat pozici zařízení v prostoru.

4.2 SteamVR tracking 2.0

Společnost Valve později uvedla technologii SteamVR tracking 2.0, která má za primární cíl snížit náklady potřebné na výrobu zařízení a zároveň vylepšit předchozí verzi. Druhá verze je založena na nových senzorech TS4231 vyráběných též firmou Triad Semiconductor. Tyto nové senzory vyžadují pro svůj chod pouze 5 externích součástek, zatímco předchozí verze vyžadovala 11 externích součástek. To má pozitivní vliv na snížení výrobních nákladů. Dále v těchto nových senzorech přibyl režim pro šetření energie, který umožňuje snížit energetickou spotřebu. Dále také přibyl nový DATA výstupní pin. Díky tomu nové senzory dokáží poslat složitější data centrální jednotce pro zpracování oproti předchozí generaci, kde TS3633 komunikovaly s centrální jednotkou pouze skrze jeden pin. Nové senzory tak nově umí dekódovat data, která byla modulována do IR záření a poslat je pak centrální jednotce jako digitální výstup skrze nový DATA pin. Vzhledem k novinkám v senzorech TS4231 byly optimalizovány i Lighthousy. Nové Lighthousy obsahují pouze jeden motor namísto dvou. Díky mechanickému inženýrství bylo možné docílit, aby jeden motor dokázal otáčet s paprsky v obou směrech – horizontálním a

vertikálním. Toto vylepšení značně sníží náklady na výrobu a tím i výslednou cenu celého zařízení. Dále se v nových Lighthousech už nenachází resetovací IR LED diody, protože se staly zbytečnými. Díky novým senzorům, které dokáží dekódovat data obsažená v laserovém paprsku, je možné do laserového paprsku zakódovat, ve kterém úhlu je paprsek aktuálně vržený do prostoru. Odpadá tak nutnost přepočtu času na úhel, ve kterém dopadl laserový paprsek na senzor. Nové senzory TS4231 jsou zpětně kompatibilní s Lighthouse 1.0, zatímco senzory TS3633 s novými Lighthousy kompatibilní nejsou. Přehled kompatibilit v rámci technologie SteamVR tracking je na obrázku 4.



Obrázek 4 Přehled kompatibility napříč verzemi technologie SteamVR tracking a zařízeními. Zdroj [9]

Díky výše zmíněným novinkám vyjde celá konstrukce jak Lighthousů, tak i 3D trackerů výrazně levněji. Dále díky odstranění resetovacích IR LED diod v Lighthousu dochází ke snížení IR interferencí a HTC Vive je tak stabilnější. Ve druhé verzi je možné využívat více než dvě stanice Lighthouse oproti první verzi, protože je možné do laserového paprsku zakódovat ID vysílací stanice. Při použití čtyř Lighthouse stanic je možné trackovat zařízení v oblasti 10x10 m.

4.3 Lighthouse

Jak již bylo zmíněno, jedná se o zařízení používající se pro trackování 3D pozice. Lighthouse bývá též označována jako Base Station. Pro trackování 3D pozice se typicky využívají dvě stanice Lightouse, které bývají nainstalovány na stěnách tak, aby byly proti sobě a měli tak na sebe výhled. Lighthouse 1.0 obsahuje dva motory, které jsou umístěny proti sobě o 90°, přičemž na každém rotoru motoru je umístěna Fresnelova čočka. Fresnelova čočka má za úkol transformovat laserový infračervený paprsek na rovinu, který je vrhán na čočku zrcadlem. Dále je ještě v Lighthouse 1.0 pole IR LED diod a fotodioda pro snímání IR paprsků z druhé Lighthouse. Právě uvedené komponenty Lighthouse 1.0 je možné vidět na obrázku 5.

Obrázek 5 Pohled na rozmontovanou Lighthouse 1.0. Zdroj [10]

Každý motor uvnitř Lighthouse rotuje konstantní rychlostí 60 Hz, což znamená, že jedna otáčka motoru trvá 16,666 ms. Mezi motory je fázový posun o 180°. Prakticky to znamená, že nejpozději za 16,666 ms jsou určeny oba úhly mezi Lighthouse a senzory. Délka jednoho cyklu tak trvá 8,333 ms. Vždy, když čočka rotoru protne značku označující 0°, pole IR LED je rozsvíceno na specifickou dobu. V délce rozsvícení IR LED jsou zakódovány 3 bity informací. V tabulce 1 jsou uvedeny délky pulzů a k nim odpovídající bitové reprezentace.

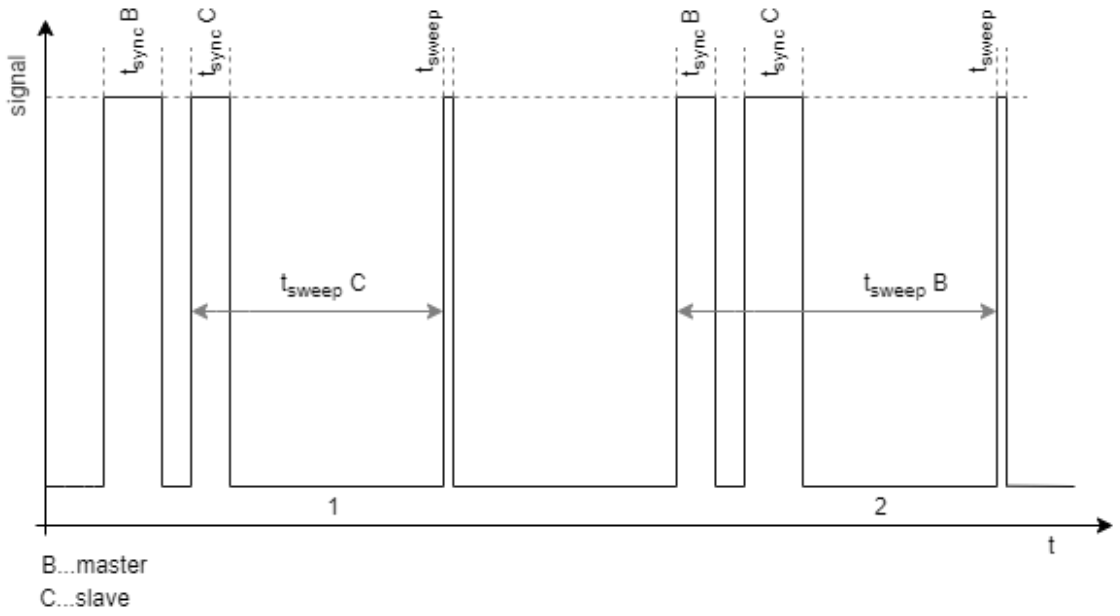
Tabulka 1 Kódování 3 bitů do délek synchronizačních pulzů Lighthouse stanic

Délka pulzu [µs]	Skip	Data	Osa
62,5	0	0	0
72,9	0	0	1
83,3	0	1	0
93,8	0	1	1
104	1	0	0
115	1	0	1
125	1	1	0
135	1	1	1

Zdroj [11]

Bit skip určuje, zdali aktuální motor vynechá aktuální cyklus, nebo bude aktivní. Bit osa určuje, který z motorů je aktuálně aktivní – horizontální 0, vertikální 1. V bitu data je zakódován 1 bit tzv. OOTX rámce, který je přenášen Lighthousem. V tomto rámci jsou přenášeny dodatečné informace o každé stanici. Více o OOTX rámcích se lze dočíst v [11, 12].

Typický průběh signálu generovaný senzorem zachytávajícím IR paprsky z Lighthouse stanic je možné vidět na obrázku 6. Obrázek 6 znázorňuje případ, kdy jsou aktivní dvě Lighthouse stanice. Písmeno B označuje master Lighthouse a písmeno C slave Lighthouse. Každý cyklus vždy začne synchronizačním pulzem master Lighthouse B, poté následuje synchronizační pulz slave Lighthouse C. Délky synchronizačních pulzů t_{syncB} a t_{syncC} udávají zakódované 3 bity, viz. Tabulka 1. Poté následuje krátký pulz, který je způsoben zásahem senzoru laserem. Časy t_{sweepB} a t_{sweepC} se odvíjí od času nástupné hrany příslušného synchronizačního pulzu, tj. pro master Lighthouse B je čas t_{sweepB} určen od nástupné hrany synchronizačního pulzu t_{syncB}.



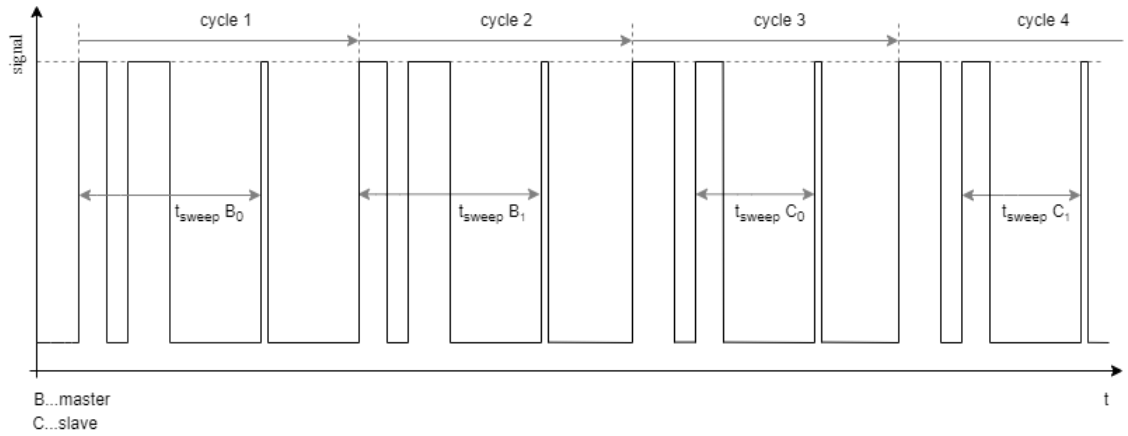
Obrázek 6 Průběh signálu generovaný senzorem zachytávajícím IR pulzy z Lighthouse stanic. Zdroj vlastní tvorba podle [13]

Časy t_{sweepB} a t_{sweepC} se použijí pro přímý výpočet úhlů, na kterých leží senzor od Lighthouse stanic. Výpočet je založen na úvaze, že motor se otáčí konstantní rychlostí a půl otáčky (180°) provede přibližně za 8,333 ms. Výpočet vertikálního úhlu θ a horizontálního úhlu φ je možný podle vzorců

$$\theta = t_{sweepV} \,/\, 0{,}008333 * \pi, \tag{1}$$

$$\varphi = t_{sweepH} \,/\, 0{,}008333 * \pi. \tag{2}$$

Jeden úhel je tedy určen vždy za jeden cyklus trvající 8,333 ms. Pro výpočet pozice senzoru jsou však nutné čtyři úhly ze dvou Lighthouse stanic. Tímto je dána obnovovací frekvence systému 30 Hz, s jakou je obnovována pozice senzoru v ideálním případě, kdy senzor nevynechá žádný laserový pulz vlivem zastínění. Celý cyklus, potřebný pro určení pozice, sestávající se ze čtyř cyklů je možné vidět na obrázku 7.



Obrázek 7 Celý cyklus potřebný pro získání čtyř úhlů ze dvou Lighthouse stanic pro výpočet pozice. Zdroj vlastní tvorba podle [13]

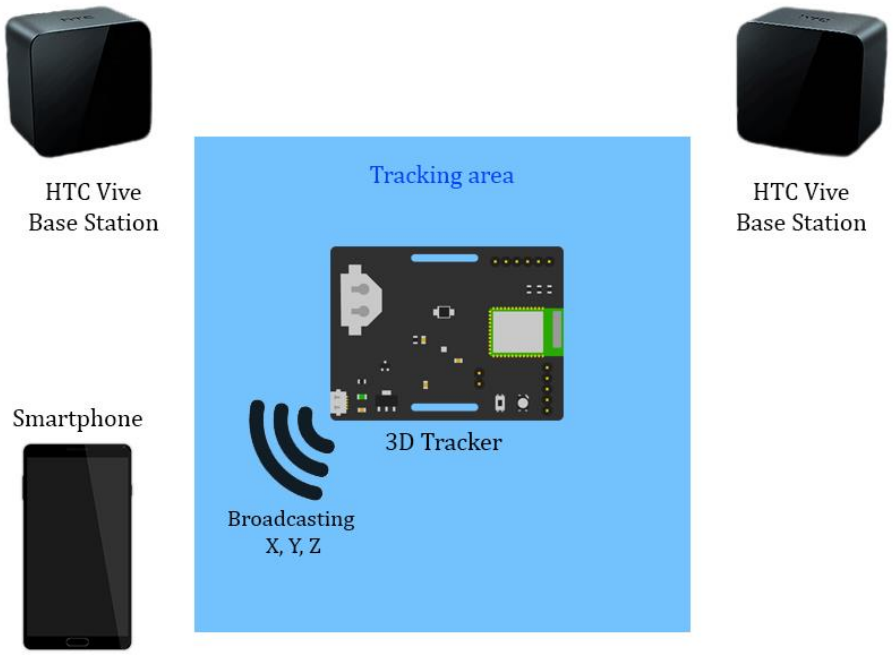
Na výše uvedeném obrázku je možné vidět typický výstup ze senzoru v případě dvou aktivních Lighthouse stanic. Celý cyklus začíná nejprve master Lighthouse, kdy je určen jako první úhel horizontální, cyklus 1. Následně je určen ve 2. cyklu i druhý úhel master Lighthouse, tedy úhel vertikální. Potom přejde master Lighthouse do skip režimu a slave Lighthouse tak přijde na řadu. Slave Lighthouse vrhá do místnosti laserové paprsky v úhlech ve stejném pořadí, jako master Lighthouse. Tedy nejprve je určen horizontální úhel v cyklu 3 a následně v cyklu 4 úhel vertikální. Pokud celý cyklus proběhl v pořádku a byly senzorem získány 4 úhly z obou Lighthouse stanic, je možné vypočítat relativní pozici senzoru v prostoru vůči Lighthouse stanicím. Kapitola Lighthouse byla zpracována podle [13].

5 Analýza a návrh řešení vlastního trackeru

5.1 Analýza

Snahou práce je vytvořit zařízení s co nejnižšími náklady pro 3D tracking. Toto zařízení by mělo být kompatibilní s HTC Vive. Je tedy nezbytné, aby nově navrhované zařízení využívalo technologii, která je použita v HTC Vive. Hlavním záměrem je tedy vytvořit zařízení podobné HTC Vive headsetu či HTC Vive VR ovladačům využívající pro 3D tracking HTC Vive Base Station (Lighthouse). Bude se jednat o malé přenosné a nízkoenergeticky náročné zařízení. Data o své poloze poté zařízení bude propagovat skrze technologii Bluetooth Low Energy.

Aby nově vytvořené zařízení mělo význam, je potřeba ještě další stěžejní komponenty systému. Je tedy potřeba ještě konzumenta, který data o poloze 3D trackeru rozumnou formou zpracuje. Tímto konzumentem se stane mobilní aplikace určená primárně pro operační systém Android. V rámci této aplikace bude uživateli vykreslena aktuální poloha 3D trackeru v prostoru. Znázornění všech komponent potřebných pro 3D tracking je na obrázku 8.



Obrázek 8 Znázornění všech komponent potřebných pro 3D tracking. Zdroj vlastní tvorba podle [14, 15]

Vzhledem k požadavku na vytvoření zařízení pro 3D tracking s co nejnižšími náklady, bylo rozhodnuto o využití pouze jednoho senzoru pro zachytávání IR pulzů z Lighthouse stanic. Tímto požadavkem je třeba si uvědomit zásadní rozdíly mezi vytvořeným 3D trackerem a zařízeními od HTC Vive. Jak uvádí zdroj [16], jenom na headsetu HTC Vive je umístěno až 32 senzorů. Lze tedy předpokládat, že přesnost trackování pozice 3D trackerem bude při použití pouze jednoho senzoru značně nižší. Aby bylo možné vypočítat relativní pozici senzoru v prostoru vůči Lighthouse stanicím, je nutné znát pozice Lighthouse stanic v prostoru. Zařízení HTC Vive získávají pozice Lighthouse stanic skrze kalibraci místnosti ve SteamVR. Takováto kalibrace je možná, protože zařízení HTC Vive obsahují mnoho senzorů, mezi kterými jsou známy vzdálenosti a jejich rozmístění na trackovaném objektu. Jakmile jsou tyto vzdálenosti známy, je poté relativně jednoduché najít, v jaké vzdálenosti od každého senzoru leží střed Lighthouse stanice. Avšak v případě navrhovaného 3D trackeru je tento problém neřešitelný, neboť vzdálenost mezi senzorem a Lighthouse stanicí na vektoru může být libovolná. Dále v případě použití pouze jednoho senzoru je nemožné získávat rotaci 3D trackeru v prostoru.

Výpočet pozice 3D trackeru v prostoru bude prováděn na základě postupu uvedeného v [17]. Výpočet je rozdělen do 5 kroků. V prvním kroku jsou nejprve vypočítány úhly mezi senzorem a Lighthouse stanicemi podle vzorců

$$\theta = \left(t_{sweepV} - 4000\right) * \pi \,/\, 8333, \tag{3}$$

$$\varphi = \left(t_{sweepH} - 4000\right) * \pi \,/\, 8333, \tag{4}$$

kde úhel θ je vertikální a úhel φ je horizontální. Hodnota 4000 je střední časový offset v μs a hodnota 8333 je délka cyklu v μs. Hodnoty t_{sweepV} a t_{sweepH} jsou doby zachycené MCU mezi začátky synchronizačních pulzů a zásahem senzoru laserem.

Ve druhém kroku jsou vypočteny normálové vektory pro obě 3D roviny Base Station – vertikální a horizontální. Souřadnice normálových vektorů jsou vypočteny podle sinových a kosinových rotačních vzorců jako

$$\vec{v} = [0; \cos(\theta)\,; \sin(\theta)], \tag{5}$$

$$\vec{h} = [\cos(\varphi)\,; 0;\ -\sin(\varphi)]. \tag{6}$$

Ve třetím kroku je poté vypočten vektor mezi Base Station a senzorem. Při výpočtu je využíváno faktu, že vektor $\vec{u}$ směřující od počátku Base Station k senzoru musí ležet na obou rovinách, což znamená, že vektor $\vec{u}$ musí být kolmý na vektory $\vec{v}$ a $\vec{h}$. Pro výpočet vektoru $\vec{u}$ lze proto použít vektorový součin vektorů $\vec{v}$ a $\vec{h}$ podle vzorce

$$\vec{u} = \vec{v} \times \vec{h}\,. \tag{7}$$

Stejný výpočet je nutné udělat také pro vektor z druhé Base Station, nechť tento vektor je označený jako $\vec{t}$.

Ve čtvrtém kroku jsou poté převedeny vypočtené vektory $\vec{u}$ a $\vec{t}$ z lokálního souřadného systému do globálního, aby bylo možné najít průsečík těchto vektorů. Konverze z lokálního souřadného systému do globálního je provedena vynásobením vektorů $\vec{u}$ a $\vec{t}$ rotačními maticemi M_1, M_2 Lighthouse stanic. Výpočet je proveden jako

$$\overrightarrow{u'} = M_1 * \vec{u}, \tag{8}$$

$$\overrightarrow{t'} = M_2 * \vec{t}. \tag{9}$$

V posledním kroku je poté vypočítán průsečík vektorů $\overrightarrow{u'}$ a $\overrightarrow{t'}$ pomocí obecného algoritmu popsaného v [18]. V tomto algoritmu se hledají dva body na dvou různoběžných přímkách, které mají mezi sebou nejkratší vzdálenost. Výhodou tohoto algoritmu je, že funguje pro jakýkoliv n dimenzionální prostor. Algoritmus předpokládá dvě rovnice přímek

$$P(s) = P_0 + s * \overrightarrow{u'}, \tag{10}$$

$$Q(t) = P_1 + t * \overrightarrow{v'}, \tag{11}$$

kde P_0, P_1 jsou počátky přímek, tj. středy Lighthouse stanic, s, t jsou neznámé skaláry a $\overrightarrow{u'}, \overrightarrow{t'}$ jsou vektory přímek, tj. vektory mezi Lighthouse stanicemi a senzorem. Nejprve je nalezena vzdálenost mezi počátky P_0, P_1 jako

$$W_0 = P_0 - P_1. \tag{12}$$

Následuje výpočet pěti skalárních součinů a, b, c, d, e podle vzorců

$$a = \overrightarrow{u'} \cdot \overrightarrow{u'}, \tag{13}$$

$$b = \overrightarrow{u'} \cdot \overrightarrow{v'}, \tag{14}$$

$$c = \overrightarrow{v'} \cdot \overrightarrow{v'}, \tag{15}$$

$$d = \overrightarrow{u'} \cdot W_0, \tag{16}$$

$$e = \overrightarrow{v'} \cdot W_0. \tag{17}$$

Poté jsou vypočítány neznámé skaláry s, t podle vzorců

$$s = \frac{be - cd}{ac - b^2}\ , \tag{18}$$

$$t = \frac{ae - bd}{ac - b^2}\ . \tag{19}$$

Jakmile jsou známy skaláry s, t, je možné dosadit do předpokládaných rovnic přímek a tím tak vypočítat body ležící na přímkách, které mají mezi sebou nejkratší vzdálenost. Výpočet je tedy proveden jako

$$P(s) = P_0 + s * \overrightarrow{u'}, \tag{20}$$

$$Q(t) = P_1 + t * \overrightarrow{v'}. \tag{21}$$

Konečná pozice 3D trackeru je stanovena jako střední bod R mezi body $P(s), Q(t)$ vypočtený jako

$$R = \frac{P(s) + Q(t)}{2}\ . \tag{22}$$

Pro výše uvedený výpočet pozice 3D trackeru je tedy potřeba znát, kde v prostoru se nachází každá Base Station a zároveň pro každou Base Station její rotační matici. Jak bylo výše řečeno, vytvořený 3D tracker nebude schopný provést kalibraci Lighthouse stanic a tím tak provést výpočet pozic a rotačních matic Lighthouse stanic. Bude tak nutné tyto informace poslat do 3D trackeru. Pro získání pozic a rotačních matic Lighthouse stanic bude využito dat získaných při kalibraci místnosti skrze SteamVR za použití zařízení HTC Vive. Jakmile bude takto kalibrován počátek v prostoru, pozice a rotační matice Lighthouse stanic, bude využito utility [19], která do konzole vypíše pro každou Base Station její pozici a rotační matici. Tyto informace budou dále označovány jako geometrie. Geometrii tak bude nutné odeslat skrze mobilní aplikaci do 3D trackerů.

5.2 Návrh řešení

Jak již bylo nastíněno v předešlé kapitole, celý systém budou tvořit celkem čtyři komponenty. Jedná se o tyto komponenty: vlastní 3D tracker, mobilní aplikace, BLE komunikace a Base Stations. Protože celé řešení bude využívat originální Base Stations vyrobené firmou HTC, bude v této kapitole popsán návrh pouze zbylých třech komponent.

5.2.1 Návrh vlastního 3D trackeru

Požadavky na vlastní 3D tracker jsou následující. Zařízení by mělo mít malé rozměry a mělo by být uzpůsobeno tak, aby bylo možné jej nosit například jako náramek. Dalším požadavkem by měla být co nejnižší energetická náročnost, avšak vzhledem k nutnosti častého přenášení dat skrze BLE se tohoto požadavku bude dosahovat obtížně. Posledním důležitým požadavkem je cena zařízení, která by opět měla být co nejnižší. Ovšem i v tomto případě je tento požadavek komplikován z důvodu drahého poštovného z USA. Kompletní návrh elektrického schématu a DPS (Deska Plošných Spojů) byl realizován v CAD nástroji EAGLE, který ve freeware verzi umožňuje návrh dvouvrstvých DPS s obsahem do 80 cm^2.

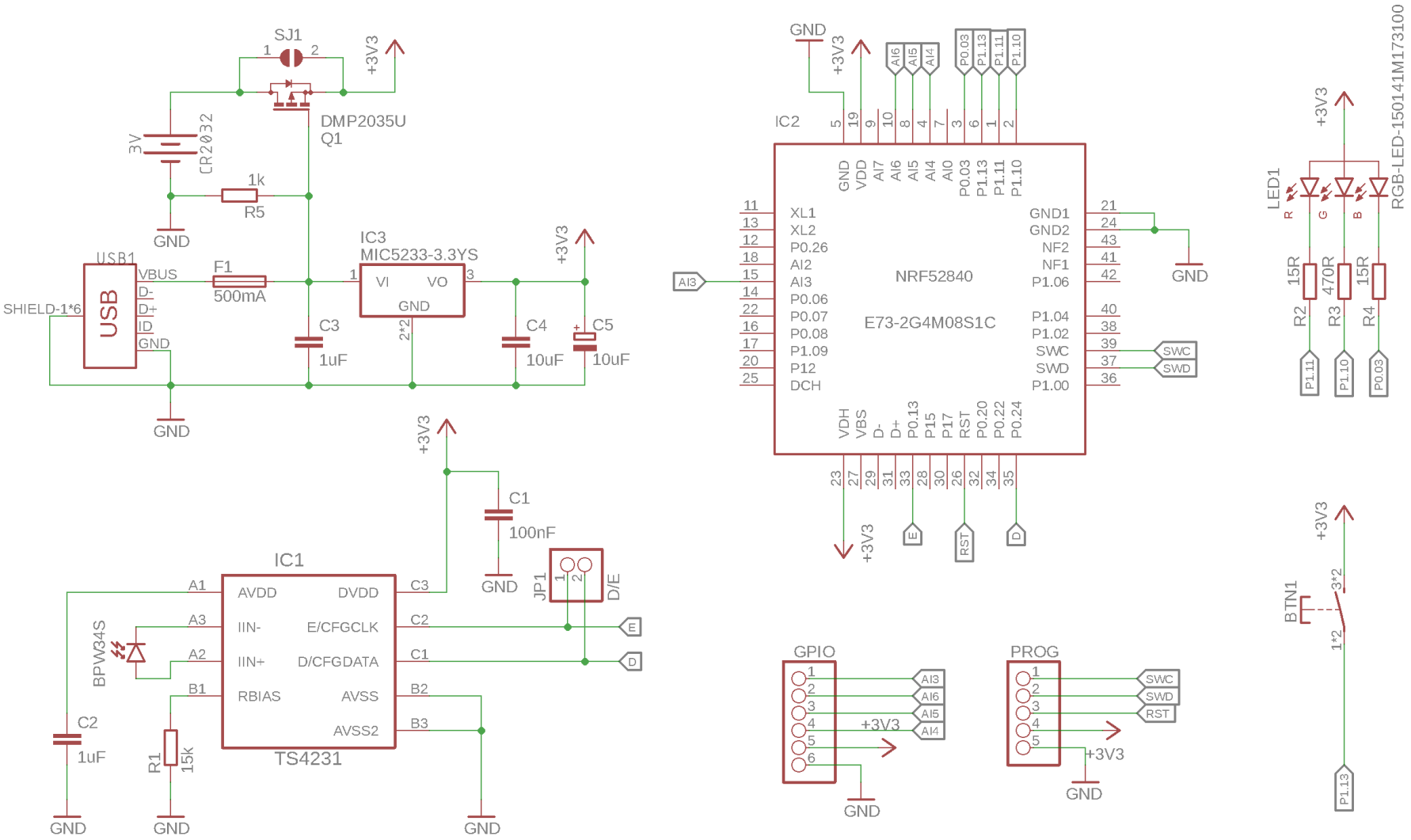
5.2.1.1 Elektrické schéma

Pro řízení celého 3D trackeru byl zvolen mikrokontroler nRF52840, který byl představen v kapitole 3.2.1. Toto MCU bylo zvoleno jednak proto, že se jedná o MCU umožňující bezdrátovou komunikaci skrze Bluetooth a zároveň proto, že se jedná již o pokročilé MCU nabízející velké možnosti. Tím je také zajištěno případné budoucí vylepšování celého zařízení.

Celý senzor pro zachytávání IR paprsků vysílaných Base Station se sestává ze dvou komponent. První komponentou je fotodioda zachytávající IR pulzy o určitém rozsahu vlnových délek. Zvolena byla doporučovaná fotodioda BPW34S. Jedná se o rychlou fotodiodu zachytávající IR pulzy o vlnové délce 430 – 1100 nm s nejvyšší citlivostí okolo 900 nm. Druhou komponentu tvoří převodník IR pulzů na digitální signál. Zvolen byl integrovaný obvod TS4231 vyráběný firmou Triad Semicondutor pro SteamVR tracking. Jedná se o následníka původního TS3633, který podporuje Steam VR tracking 2.0 a zároveň je zpětně kompatibilní s Base Stations verze 1.0.

Aby mohlo být zařízení přenositelné, je napájeno z jedné lithiové 3 V baterie CR2032. Protože není jisté, jak bude výsledné zařízení energeticky náročné, byla též přidána možnost napájet tracker pomocí mikro USB konektoru. Díky tomu bude možné připojit tracker například na powerbanku a získat tak daleko delší provozní dobu. Dalším důvodem přidání alternativního zdroje napájení bylo pro zjednodušení vývoje.

Dále byla trackeru přidána jedna RGB LED dioda a jedno tlačítko umožňující komunikaci 3D trackeru s uživatelem. Při výběru RGB LED diody bylo dbáno převážně na co nejnižší úbytky napětí v propustném směru pro každou diodu, a to z důvodu, aby byly diody schopné pracovat co nejdéle při napájení z baterie. Kompletní elektrické schéma je uvedeno na obrázku 9.



Obrázek 9 Elektrické schéma navrhovaného 3D trackeru. Zdroj: autor

Nyní bude popsáno celé schéma uvedené na obrázku č. 9. Napájecí část byla navržena tak, aby bylo možné tracker napájet jak z baterie CR2032, tak i pomocí mikro USB konektoru. Toto zapojení disponuje ochranou proti přepólování baterie a tím tak ochrání zbytek obvodu před poškozením. Ochranu proti přepólování zde zajišťuje tranzistor P-MOSFET (Metal Oxid Semiconductor Field Effect Tranzistor) *Q1*. Je využíváno jevu, kdy P-MOSFET tranzistor je vodivý tehdy, je-li rozdíl napětí mezi elektrodou Source a Gate záporný, elektrická vlastnost tranzistoru V_{GS}. Toho je docíleno běžně tak, že elektroda Gate je připojena na záporný pól napájecího zdroje, tzv. zem. Ve chvíli, kdy je baterie připojena opačnou polaritou, kdy se na elektrodu Gate připojí kladné napětí, není možné, aby vznikl záporný rozdíl napětí na elektrodách Source a Gate a obvod tak zůstane rozpojen. Další příjemnou vlastností při využívání P-MOSFET tranzistoru, jako ochrany proti přepólování napájecího zdroje, je jeho velice nízký odpor – elektrická vlastnost $R_{DS(ON)}$, jejíž hodnoty jsou běžně desítky mΩ. Díky tomu je možné efektivně chránit obvod proti přepólování bez zbytečných ztrát energie, která by byla parazitně odebírána při provozu zařízení, což je při provozu z baterie velice nevhodné. Tranzistor *Q1* byl vybrán pro nízkou hodnotu $R_{DS(ON)}$ a nízká napětí V_{GS}.

Pokud je tracker napájen pomocí USB konektoru, je postupně přivedeno kladné napětí 5 V skrze pojistku *F1* na pin 1 *IC3* a elektrodu Gate tranzistoru *Q1*. Pojistka *F1* je resetovatelná pojistka dimenzovaná na proud 500 mA z důvodu, kdy standartně USB 2.0 port v počítači je schopen dodat proud do 500 mA. Kdyby se nešťastnou náhodou vytvořil zkrat na desce trackeru, tato pojistka by měla ochránit USB port před zničením. Přivedené napětí na elektrodu Gate tranzistoru *Q1* způsobí, že se tranzistor *Q1* uzavře a dojde tak k odpojení baterie od obvodu. Rezistor *R5* připojuje elektrodu Gate tranzistoru *Q1* k zápornému pólu zdroje. To je důležité zejména v případě, kdy je k obvodu připojena pouze baterie. Dále rezistor *R5* připojuje k zápornému pólu napájecího zdroje pin 1 *IC3* a umožňuje vybíjení kondenzátoru *C3*. Integrovaný obvod *IC3* je LDO (Low-DropOut) regulátor, který snižuje napětí z přivedených 5 V na 3,3 V. Toto snížení napájecího napětí je nutné, protože integrovaný obvod TS4231 vyžaduje napětí 3,3 V. Z tohoto důvodu je napětí na napájecí sběrnici 3,3 V. Regulátor napětí *IC3* je vybaven ochranou proti tekoucímu zápornému proudu z výstupu LDO na vstup. Tato ochrana je aktivována

ve chvíli, kdy vstupní pin 1 *IC3* je připojen k zápornému pólu napájecího zdroje. Jakmile dojde k odpojení napětí 5 V z mikro USB konektoru, rezistorem *R5* se začne vybíjet kondenzátor *C3*. Na základě velikosti proudu potřebného pro zcela vybití kondenzátoru *C3* byla určena hodnota rezistoru *R5* na hodnotu 1 kΩ. Pokud by odpor rezistoru *R5* byl příliš velký, vybíjecí proud kondenzátoru by nemusel být dostatečný a mohlo by se stát, že tranzistor *Q1* by zůstal pouze pootevřený a skrze výstup *IC3* by tekl záporný proud na vstupní pin 1 *IC3*. Tento proud by dobíjel kondenzátor *C3*, přizavíral tranzistor *Q1* a tekl skrze rezistor *R5* k zápornému pólu napájecího zdroje. Kondenzátory *C3* a *C4* zde slouží pro zamezení kmitání *IC3* a kondenzátor *C5* je zde uveden pro případ, kdy by nestačila kapacita 10 µF keramického kondenzátoru *C4* pro pokrytí špičkových proudových odběrů MCU, například vlivem vysílání Bluetooth. K tranzistoru *Q1* je ještě paralelně připojená propojka *SJ1*, tzv. solder bridge, která umožňuje neosazovat součástky *Q1*, *R5*, *USB1*, *F1*, *C3*, *IC3* a tím tak zlevnit výrobu trackeru. Nicméně jeden z kondenzátorů *C4* nebo *C5* by stejně měl být osazen. Poté se při výrobě cínem propojí propojka *SJ1*. Toto zapojení napájení tedy umožňuje automatické přepínání mezi napájením z baterie a mikro USB konektoru.

Jak již bylo zmíněno, senzor zachytávající IR pulzy je tvořen fotodiodou a převodníkem IR pulzů na digitální signál. Integrovaný obvod TS4231 byl zapojen dle katalogového zapojení. Kondenzátor *C1* je zde typicky zapojen s hodnotou 100 nF, aby plnil funkci blokovacího kondenzátoru. Blokovací kondenzátory se běžně připojují k různým integrovaným obvodům, protože například při rychlých změnách logických úrovní dochází k impulzní potřebě energie, která by jinak tekla od zdroje příliš dlouho. TS4231 komunikuje s MCU skrze dva piny – *D* a *E*. Tyto piny jsou tedy připojeny k nRF52840, přičemž jsou paralelně připojeny k pinheaderu. Pinheader je zde proto, aby bylo možné připojit piny *D* a *E* k osciloskopu a sledovat tak průběhy signálů vysílané TS4231.

K nRF52840 jsou dále připojeny ještě další 2 pinheadery. První pinheader označený jako *PROG* slouží pro připojení programátoru, kterým je nahrán kód firmwaru přímo do flash paměti nRF52840. Druhým piheaderem je pinheader označený jako *GPIO*. Skrze tento pinheader jsou vyvedeny další vstupně/výstupní piny nRF52840, které mohou být využity pro připojení externích periferií.

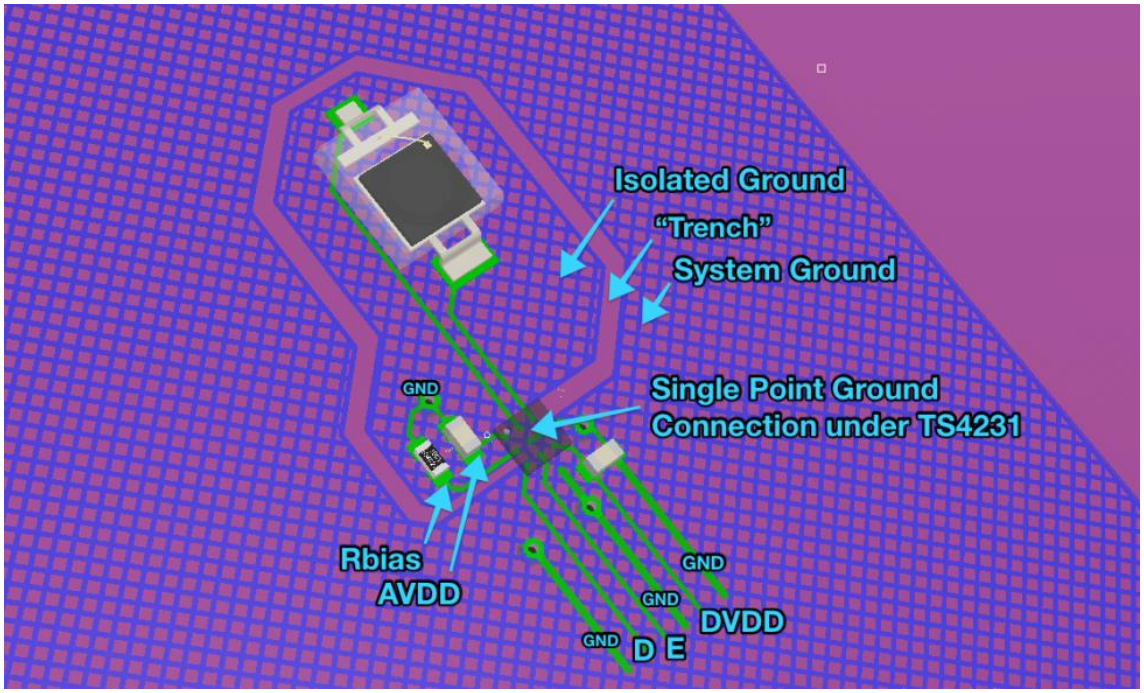
Dále je k nRF52840 zapojena RGB LED dioda a jedno tlačítko. Každá dioda je připojena k MCU skrze předřadný rezistor, který omezuje proud tekoucí diodou. Hodnoty předřadných rezistorů byly zvoleny tak, aby jednotlivé barvy RGB LED diody dosahovaly podobné svítivosti. Tlačítko je připojeno k nRF52840 napřímo a bude využito interního pull-down[3] rezistoru.

Pro tracker není využíváno samostatného MCU nRF52840 jak se může zdát ze schematické značky, ale je využito bezdrátového modulu E73-2G4M08S1C vyrobeného firmou EBYTE. Tento modul obsahuje nRF52840 a je v něm obsažené zapojení, které je potřeba, aby vůbec samotný nRF52840 mohl pracovat. Jedná se zejména o anténu, blokovací kondenzátory, externí oscilátor a další.

5.2.1.2 Návrh DPS

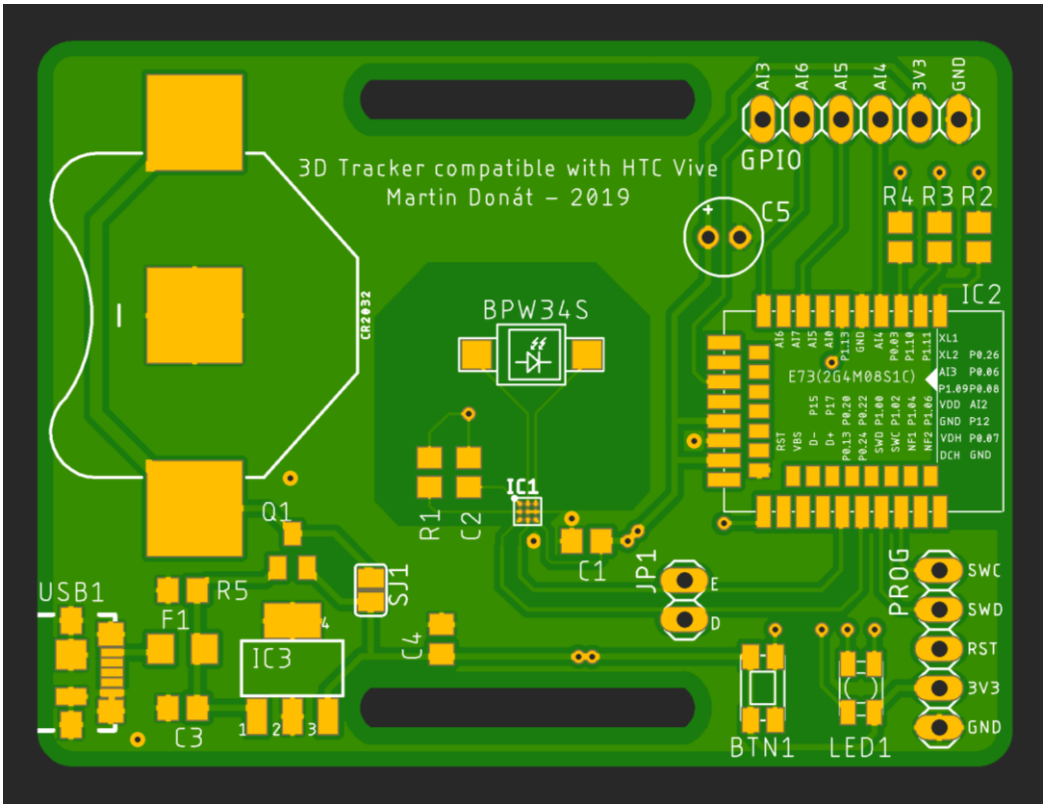
V této kapitole bude představen samotný návrh DPS 3D trackeru. Protože by se mělo jednat o malé přenosné zařízení, bylo při návrhu DPS dbáno na malé rozměry. Dále bylo dbáno na pokyny uváděné firmou Triad Semiconductor pro integrovaný obvod TS4231 uvedené v dokumentu TS4231 Design Guidelines dostupného z [20]. První doporučení uvádí, jak snížit šum a interference, které mohou vznikat na měděných spojích mezi TS4231 a fotodiodou – piny *IN+* a *IN-*. V tomto doporučení je uvedeno, aby spoje byly stejně dlouhé, stejného tvaru a dosahovaly stejných elektrických vlastností. Ve druhém doporučení je uvedeno, jak snížit parazitní kapacitanci na spojích *IN+* a *IN-*. Toho by mělo být docíleno tak, že šířka spoje by měla být minimální, například 6 – 8 mil (milliinch) a zároveň mezera mezi středy spojů by měla být široká okolo 0,5 mm. Dále by vyplňující polygony připojené k zápornému pólu zdroje neměly být celistvé, nýbrž šrafované. Poslední doporučení týkající se návrhu DPS je věnováno tomu, jak prakticky implementovat izolaci země pro TS4231 a snížit tak interference, které mohou vznikat indukcí způsobenou vracejícími se proudy skrze spoje nacházející se podél polygonů připojených k zápornému pólu zdroje. Příklad návrhu implementace izolace země pro TS4231 lze vidět na obrázku 10.

[3] Pull-down rezistor omezuje proud tekoucí pinem MCU ve chvíli, kdy je tlačítko stisknuto a zároveň nastavuje na pinu MCU nízkou logickou úroveň, když tlačítko stisknuté není.

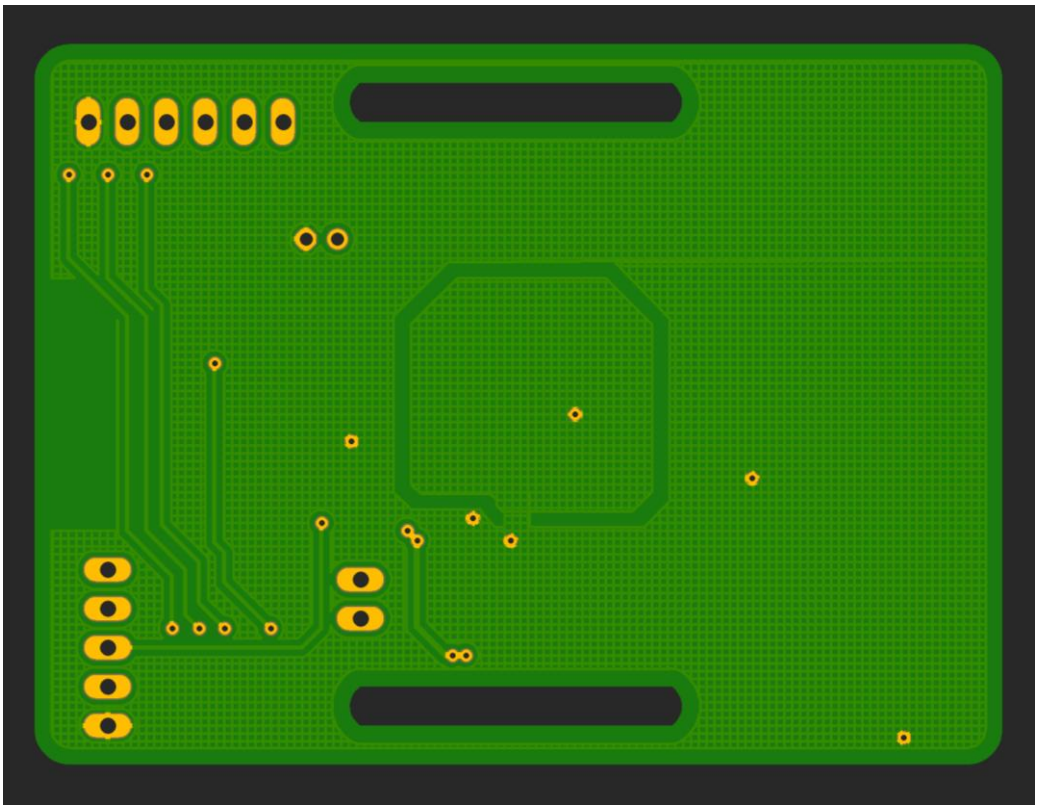


Obrázek 10 Příklad implementace izolace země pro TS4231. Zdroj [20]

Výsledná DPS byla navržena jako dvouvrstvá o rozměru 6,3x4,7 cm a veškeré volné plochy byly vylity šrafovaným polygonem připojeným k zemi. Dalším velice důležitým aspektem bylo umístění fotodiody co nejblíže středu DPS a to tak, aby fotodioda byla co nejméně stíněná okolními součástkami, neboť schopnost trackovat pozici 3D trackerem je přímo závislá na zachytávání IR pulzů fotodiodou. Aby mohl být tracker nošen například jako náramek, byly přidány dva oválné otvory o rozměru 22x2,5 mm po delších stranách DPS pro prostrčení pásku. Dále bylo nutné vynechat polygon v obou vrstvách v místě, kde se nachází anténa bezdrátového modulu E73-2G4M08S1C. Kdyby se tak nestalo, anténa by byla stíněná a tím by se výrazně snížil dosah Bluetooth, případně by Bluetooth nemuselo vysílat vůbec. Pinheadery *GPIO* a *PROG* byly umístěny na kraj DPS pro dobrou dostupnost, stejně jako RGB LED dioda, tlačítko, mikro USB konektor a držák baterie CR2032. Výsledný návrh DPS lze vidět na obrázcích 11 a 12.



Obrázek 11 Vykreslená vrchní strana navržené DPS pro 3D tracker.



Obrázek 12 Vykreslená spodní strana navržené DPS pro 3D tracker.

5.2.1.3 Kalkulace nákladů

Vzhledem k požadavku na nízkou cenu zařízení zde bude provedena kalkulace celkových nákladů. Výrobu DPS podle návrhu zajistila společnost PCBWay. Bohužel nebylo možné zakoupit integrované obvody TS4231 na tuzemském trhu a ani od zahraničních distributorů. Integrované obvody TS4231 musely být proto zakoupeny přímo od výrobce z USA při ceně $1,85 za kus. Výrobce prodává tyto obvody po 10 kusech. Vzhledem k velmi drahému poštovnému byla využita služba pro přeposlání balíku z USA do České republiky, která vyjde na značně nižší náklady. Z výše zmíněných důvodů bylo objednáno rovnou 20 kusů TS4231. Bezdrátové moduly obsahující nRF52840 byly objednány z AliExpressu. Zbylé součástky byly objednány od distributora Farnell, který je dopravil do České republiky ze skladů ve Velké Británii. Soupis součástek zakoupených pro výrobu 3D trackerů je uveden v tabulce 2.

Tabulka 2 Souhrn všech nákladů na výrobu 3D trackerů

Součástka	Hodnota	Množství	Cena v Kč (bez DPH)
-	BPW34S	5	152,8
BTN1	-	5	36,47
C1	100nF	5	10,7
C2, C3	1uF	10	24,27
C4	10uF	5	14,51
-	CR2032	5	75,37
F1	500mA	5	18,62
LED1	150141M173100	5	65,23
Q1	DMP2035U	5	28,25
R1	15k	10	1,69
R2, R4	15R	10	1,59
R3	470R	10	1,69
R5	100k	10	1,67
USB1	Micro-Type-B	5	97,84
IC3	MIC5233-3.3YS	5	84,36
Doprava Farnell	-	-	130

DPH k součástkám		-	156,46
Součástka	**Hodnota**	**Množství**	**Cena v Kč (s DPH)**
IC1	TS4231	20	883,8
IC2	E73-2G4M08S1C	2	308,54
Výroba DPS		5	142,66
Doprava po USA		-	188,7
Doprava z USA do ČR		-	833,55
Celková suma		**-**	**3258,77**

Na výrobu 3D trackerů se celková částka vyšplhala na 3259 Kč. Zprvu se počítalo s výrobou dvou 3D trackerů, a proto byly objednány pouze dva bezdrátové moduly E73-2G4M08S1C. Poté bylo zjištěno, že velikost integrovaných obvodů TS4231 je příliš malá a vyrobit tak DPS v domácích podmínkách by bylo téměř nemožné. Bylo proto rozhodnuto, že se DPS nechají vyrobit u profesionálního výrobce. Všichni profesionální výrobci vyrábějí DPS minimálně po pěti kusech. Nakonec byl zvolen výrobce PCBWay, který nabízel výrobu za příznivou cenu. Zbylé součástky byly poté objednány z Farnellu v takovém množství umožňujícím výrobu pěti 3D trackerů.

Byla provedena kalkulace nákladů, kolik by teoreticky stála výroba jednoho 3D trackeru bez zahrnutí veškerých vedlejších nákladů. V tomto teoretickém výpočtu bylo počítáno s přesným množstvím potřebných součástek, které DPS vyžaduje pro osazení. Rovněž byla uvažována cena za výrobu jedné DPS. Výslednou kalkulaci lze vidět v tabulce 3.

Tabulka 3 Kalkulace nákladů na výrobu jednoho 3D trackeru nezahrnující přepravní náklady

Součástka	Hodnota	Množství	Cena v Kč (bez DPH)
-	BPW34S	1	30,56
BTN1	-	1	7,29
C1	100nF	1	2,14
C2, C3	1uF	2	4,85
C4	10uF	1	2,9

-	CR2032	1	15,07
F1	500mA	1	3,72
LED1	150141M173100	1	13,05
Q1	DMP2035U	1	5,65
R1	15k	1	0,17
R2, R4	15R	2	0,32
R3	470R	1	0,17
R5	100k	1	0,17
USB1	Micro-Type-B	1	19,57
IC3	MIC5233-3.3YS	1	16,87
DPH k součástkám		-	25,73
Součástka	**Hodnota**	**Množství**	**Cena v Kč (s DPH)**
IC1	TS4231	1	44,19
IC2	E73-2G4M08S1C	1	154,27
Výroba DPS		1	28,53
Celková suma		**-**	**375,22**

5.2.2 Návrh mobilní aplikace

Jak již bylo zmíněno výše, mobilní aplikace je důležitou součástí systému, bez které by samotný 3D tracker nenabýval příliš na významu. V rámci této kapitoly bude proto představen návrh mobilní aplikace. Nejprve budou vyloženy požadavky na samotnou aplikaci a poté zde budou uvedeny wireframes návrhy pro jednotlivé obrazovky.

Mobilní aplikace musí umožnit uživateli vyhledávat okolní 3D trackery pomocí BLE. Dále je nutné, aby aplikace umožnila uživateli navázat i ukončit BLE spojení s objevenými 3D trackery. Jakmile je spojení navázáno, je možné přijímat vypočtené souřadnice z 3D trackerů. To je možné za podmínky, že každý 3D tracker má nastavenou tzv. geometrii, která v sobě zahrnuje pozici a rotační matici pro každou Base Station. Bez geometrie není možné, aby 3D tracker vypočítal vlastní pozici v prostoru. Pokud tedy není geometrie v 3D trackeru nastavena, je nezbytné tuto geometrii 3D trackeru nastavit. Geometrii tak musí být možné importovat do

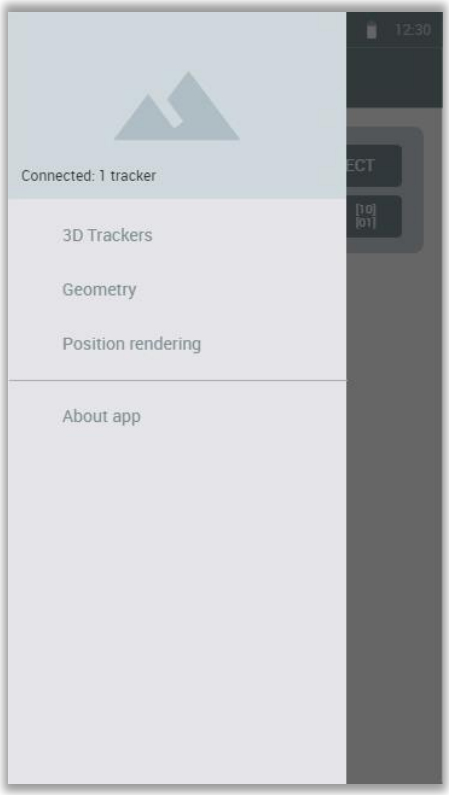
aplikace a poté skrze BLE do 3D trackeru odeslat. Protože by celé řešení, tedy 3D tracker a mobilní aplikace, mělo být využíváno převážně pro demonstrace veřejnosti na vědecko-popularizačních akcích, měla by být mobilní aplikace dostupná i pro veřejnost. Zde není příliš žádané, aby mohl kdokoliv měnit geometrii 3D trackerům a tím tak přidělávat obsluze práci. Bylo by proto vhodné, aby aplikace byla rozdělena do dvou uživatelských režimů – administrátor a běžný uživatel. V neposlední řadě je geometrie též důležitá pro vykreslování, aby bylo možné vykreslit uživateli obě Base Stations. Aplikace proto musí být schopná vyčíst aktuální geometrii z 3D trackerů srze BLE. Poslední požadavek na mobilní aplikaci se týká samotného vykreslování. Zde je kladen důraz na správnost vykreslení například týkající se realistického měřítka. Aby bylo možné dobře prohlížet pozice 3D trackerů v prostoru na 2D zobrazovacím displeji, musí být možné natáčet s kamerou a měnit přiblížení kamery. Dále je vhodné, aby aplikace umožňovala přepínání mezi tmavým a světlým pozadím. V neposlední řadě je důležité vykreslit počátek a Base Stations.

Z výše uvedených požadavků na aplikaci bylo stanoveno rozvržení aplikace na tři hlavní obrazovky. Protože operační systém Android pracuje s terminologiemi Aktivita a Fragment pro jednotlivé obrazovky, budou tak označovány i v této práci. V Androidu je možné rozdělit několik obrazovek aplikace do aktivit nebo do několika fragmentů, které jsou součástí jedné nebo více aktivit. Protože vytváření aktivit je poměrně náročné na systémové zdroje, byly proto vyvinuty fragmenty, které jsou hostované v rámci aktivity a jejich vytváření klade nižší nároky na systémové zdroje. Aplikace byla proto navržena tak, aby využívala jednu hlavní aktivitu a různé obrazovky byly rozděleny do fragmentů. Návrhy wireframes aplikace byly vytvořeny v online nástroji FluidUI, který umožňuje v bezplatné verzi vytvořit jeden projekt obsahující 10 obrazovek.

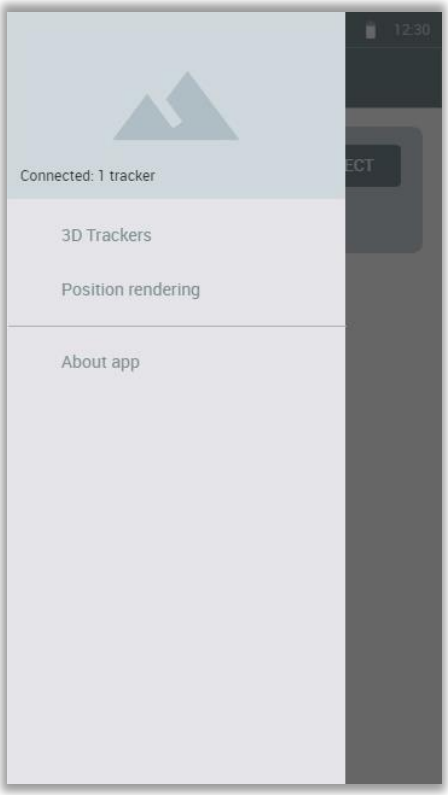
5.2.2.1 Hlavní menu aplikace

Každá aplikace sestávající se z několika obrazovek potřebuje rozumnou formou nabídnout uživateli navigaci, skrze kterou má uživatel možnost přecházet mezi jednotlivými obrazovkami. Není tomu jinak ani u této aplikace, protože jak již bylo zmíněno výše, aplikace bude rozdělena do tří fragmentů. Jedná se o následující

fragmenty: 3D trackery, Geometrie a Vykreslování pozice. Pro každý fragment tak bude přidána položka do hlavního menu aplikace. Dále bude aplikace obsahovat dialog O aplikaci, ve kterém budou uvedeny základní informace o aplikaci. Tento dialog bude též přístupný skrze hlavní menu aplikace. Protože aplikace bude rozdělena do dvou uživatelských režimů, budou zde uvedeny návrhy hlavního menu pro každý režim. Výsledné návrhy aplikačního menu je možné vidět na obrázcích 13 a 14.



Obrázek 13 Hlavní menu aplikace v administrátorském režimu.



Obrázek 14 Hlavní menu aplikace v uživatelském režimu.

Jak je patrné z výše uvedených obrázků, administrátorský režim umožní navíc pracovat s geometrií 3D trackeru. Před běžnými uživateli tak bude skrytý fragment Geometrie. Pro hlavní menu aplikace bude využito komponenty *DrawerLayout* vyvinuté společností Google pro material design. Takovéto menu se vždy animovaně vysouvá nejčastěji z levé strany a následně i zasouvá zpátky do stejné strany. Menu komponenta *DrawerLayout* je pojmenována vcelku trefně, neboť v angličtině drawer znamená šuplík. Dále je možné na výše uvedených

obrázcích aplikačního menu vidět, že v horním prostoru menu je umístěno záhlaví obsahující pozadí a text. Text je informativního charakteru, který přehledně udává uživateli, ke kolika 3D trackerům je aktuálně připojený.

5.2.2.2 Fragment 3D trackery

Bude se jednat o hlavní obrazovku, kterou uživatel uvidí hned po spuštění aplikace. V rámci tohoto fragmentu budou uživatelem spravovány 3D trackery. Uživatel bude moci kdykoliv obnovit seznam objevených zařízení skenováním okolních 3D trackerů skrze BLE popotažením obrazovky směrem dolů. Pro takovéto obnovování 3D trackerů bude využito komponenty *SwipeRefreshLayout*, která je dnes hojně využívána moderními aplikacemi a je tak intuitivní pro většinu uživatelů. Dále zde budou uživateli zobrazeny informace o každém 3D trackeru. Jedná se o informace týkající se stavu připojení, název zařízení, MAC adresa a aktuální stav 3D trackeru. Stav 3D trackeru bude uživatele informovat zejména o tom, zdali je nastavena geometrie 3D trackeru a zdali jsou aplikací přijímány souřadnice z 3D trackeru. V administrátorském režimu bude navíc oproti uživatelskému režimu u každého trackeru tlačítko s ikonou matice, které bude sloužit pro odeslání aktuální nastavené geometrie 3D trackeru. Výsledný návrh fragmentu 3D trackery lze vidět na obrázcích 15 a 16.

Obrázek 15 Fragment 3D trackery v administračním režimu.

Obrázek 16 Fragment 3D trackery v uživatelském režimu.

5.2.2.3 Fragment Geometrie

Jak již bylo zmíněno v předešlé kapitole 5.2.2.1 zabývající se návrhem hlavního menu aplikace, fragment Geometrie bude přístupný pouze v administrátorském režimu. Tento fragment umožní uživateli importovat geometrii z textového souboru do aplikace. Aplikace během importu uloží geometrii do interní paměti telefonu, odkud se jí pokusí s každým spuštěním aplikace načíst, a zároveň nastaví importovanou geometrii jako aktuální. Aktuální geometrie bude aplikací využívána pro vykreslování a odesílání geometrie 3D trackeru. S každým připojením k 3D trackeru se bude aplikace snažit nastavit přečtenou geometrii z 3D trackeru jako aktuální, v případě, že přečtená geometrie z 3D trackeru byla nastavena. Návrh fragmentu Geometrie je možné vidět na obrázku 17.

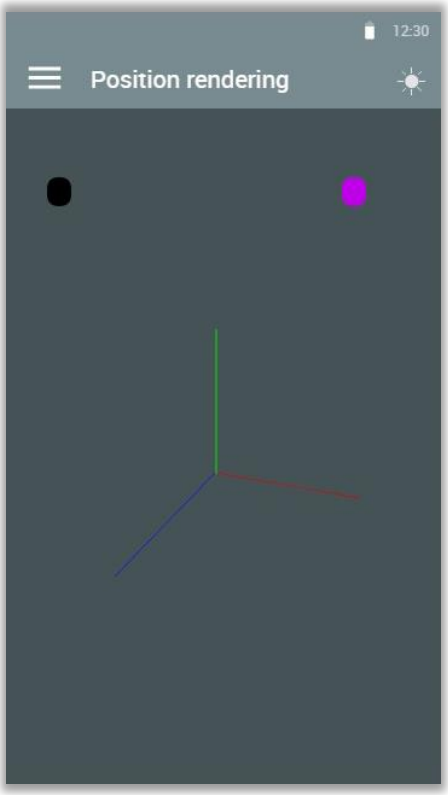
Obrázek 17 Fragment Geometrie, který je dostupný pouze v administrátorském režimu.

Importování geometrie bude iniciováno kliknutím na ikonu plus v hlavní liště aplikace. Poté uživatel skrze prohlížeč souborů vybere textový soubor s geometrií. Vzhledem k automatickému nastavování geometrie po připojení k 3D trackeru je v hlavní liště aplikace umístěno ještě druhé tlačítko s ikonou dvou šipek. Po kliknutí na toto tlačítko dojde v aplikaci k nahrazení aktuální geometrie uloženou geometrií. Tato funkce se bude hodit zejména ve chvíli, kdy bude chtít administrátor nastavit všem 3D trackerům novou geometrii. Pro každou geometrii zde budou uživateli vypsány informace pro každou Base Station, týkající se umístění Base Station v prostoru a její rotační matice.

5.2.2.4 Fragment Vykreslování pozice

Z uživatelského hlediska se bude jednat o nejstěžejnější fragment, který dodá celému systému na významu. V tomto fragmentu, jak již název vypovídá, se budou vykreslovat pozice všech připojených 3D trackerů ve 3D grafice. Celý fragment tak bude tvořit jediná view komponenta *GLSurfaceView*, do které se bude vykreslovat.

Vykreslen zde bude počátek souřadného systému, tj. souřadnice [0; 0; 0], ve kterém budou umístěny osy *x*, *y* a *z*. Dále budou ve trojrozměrné scéně vykresleny obě Base Stations na souřadnicích z aktuální geometrie a natočeny v prostoru podle rotačních matic. Nakonec budou vykresleny i všechny připojené 3D trackery na jejich souřadnicích barvou stanovenou během skenování 3D trackerů. Tato barva bude odpovídat barvě 3D trackeru ve fragmentu 3D trackery. Výsledný wireframe fragmentu Vykreslování pozice lze vidět na obrázku 18.



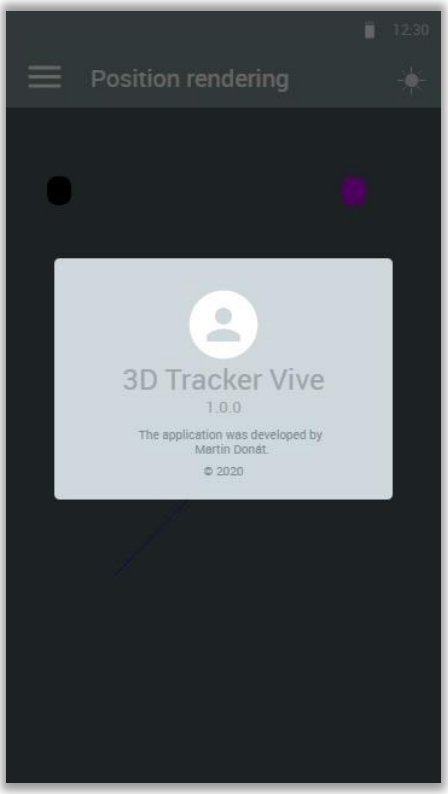
Obrázek 18 Návrh fragmentu Vykreslování pozice.

Aby mohla být věrně vykreslena pozice 3D trackeru uživateli, bude možné natáčet s kamerou okolo počátku souřadného systému a tím tak natočit celou scénu uživateli tak, aby pozice 3D trackeru byla dobře pozorovatelná. Dále bude uživateli umožněno celou scénu přibližovat a oddalovat pomocí běžně používaného gesta využívajícího dva prsty. V neposlední řadě bude uživateli umožněno přepínat mezi tmavým a světlým pozadím scény skrze kliknutí na tlačítko s ikonou slunce

umístěné v hlavní liště aplikace. Díky tomu by se měla aplikace lépe používat v prostorách, kde je hodně světla.

5.2.2.5 Dialog O aplikaci

Dialog O aplikaci bude sloužit uživatelům převážně pro zobrazování informací o aplikaci. Dialog bude obsahovat ikonu aplikace, název aplikace, informaci o aktuální verzi aplikace, autora a rok vytvoření. V tomto dialogu bude také ukryto aktivování administrátorského režimu před běžnými uživateli. Pro aktivování administrátorského režimu bude potřeba 10krát za sebou kliknout na ikonu aplikace. Jakmile se tak stane, aplikace se přepne do administrátorského režimu a bude možné využívat fragmentu Geometrie a odesílat geometrii do 3D trackerů. Navržený dialog O aplikaci je možné vidět na obrázku 19.



Obrázek 19 Návrh dialogu O aplikaci.

5.2.3 Návrh Bluetooth komunikace

Poslední komponentu celého systému tvoří komunikace mezi 3D trackerem a mobilní aplikací skrze BLE. Vzhledem k využití BLE pro komunikaci je nutné nejprve vymezit role a následně způsob předávání dat.

3D tracker bude vystupovat v roli peripheral, bude se tedy chovat jako server, který zpřístupňuje data ostatním zařízením skrze služby a charakteristiky. Smartphone s nainstalovanou aplikací bude vystupovat v roli central a bude se tak chovat jako klient, který posílá požadavky na data serveru.

Celkově bude 3D tracker nabízet dvě služby. První služba bude pojmenována jako Coordinates Service a bude obsahovat jedinou stejnojmennou charakteristiku, do které bude 3D tracker zapisovat souřadnice *x*, *y* a *z*. Tuto charakteristiku bude central zařízení moci přečíst kdykoliv bude chtít. Zároveň bude tato charakteristika implementovat mechanismus notify, skrze který budou vždy data odeslána z 3D trackeru do smartphonu, jakmile dojde k jejich změně.

Druhá služba bude zodpovědná za nastavování a zprostředkovávání geometrie. Bude tedy proto pojmenována jako Base Station Geometry Service. V této službě bude celkem pět charakteristik umožňujících zápis a čtení dat. První charakteristika označená jako Station Number bude sloužit pro zapsání indexu Base Station, který bude využit v následujících charakteristikách pro zápis a čtení dat. Indexy bude možné zapsat pouze dva, protože systém využívá právě dvou Base Station. Index 0 odpovídá geometrii pro master Base Station a index 1 pro slave Base Station. Druhá charakteristika Position bude sloužit pro nastavování pozice Base Station vždy podle nastaveného indexu v charakteristice Station Number. Následují tři charakteristiky, které umožňují zápis a čtení rotační matice po řádcích vždy též podle nastavené hodnoty indexu charakteristiky Station Number. Charakteristiky jsou pojmenovány jako Rotation Row 1, Rotation Row 2 a Rotation Row 3. Rozdělení čtení a zápisu rotační matice do třech charakteristik bylo zejména pro zjednodušení, neboť BLE standartně podporuje data o délce 20 bytů v jednom paketu. Rotační matice je čtvercová matice o rozměru 3x3, obsahuje tedy celkem 9 floatových hodnot, přičemž celková velikost rotační matice činí 36 bytů a pro její přenos skrze

jednu charakteristiku by vyžadoval určitá nastavení v BLE komunikaci. Přehled služeb a charakteristik je uveden v tabulce 4.

Tabulka 4 Přehled služeb a charakteristik poskytovaných 3D trackerem

Služba	Charakteristika	Čtení/Zápis/Notify	Rozsah hodnot	Popis
Coordinates	Coordinates	R/N	12 B	Souřadnice x, y, z 3D trackeru
Base Station Geometry	Station Number	R/W	1 B (0 – 1)	Index Base Station
	Position	R/W	12 B	Pozice Base Station
	Rotation Row 1	R/W	12 B	První řádek rotační matice Base Station
	Rotation Row 2	R/W	12 B	Druhý řádek rotační matice Base Station
	Rotation Row 3	R/W	12 B	Třetí řádek rotační matice Base Station

6 Implementace

6.1 Firmware 3D trackeru

Kompletní firmware 3D trackeru byl vyvíjen v PlatformIO. Jedná se o multiplatformní nástroj, podporující různé architektury a frameworky pro embedované systémy. Firmware byl tak napsán v programovacím jazyce C++ za použití frameworku Arduino.

Firmware 3D trackeru byl rozdělen celkem do pěti komponent. Jedná se o následující komponenty: *Bluetooth*, *PulseProcessor*, *PositionCalculator*, *TrackerGPIO* a *LedDriver*. Tyto komponenty budou v následujících kapitolách popsány.

6.1.1 Vstupní bod firmwaru (soubor main.cpp)

Programy psané v jazyce C++ začínají vždy ve funkci *main()*. Protože bylo využito frameworku Arduino, který funkci *main()* implementuje, je tímto vstupní bod programu rozdělen do dvou funkcí – *setup()* a *loop()* nacházející se uvnitř souboru main.cpp. Při startu programu je vždy nejprve zavolána jednou funkce *setup()*, která slouží pro prvotní nastavení programu. Poté je volána v nekonečném cyklu funkce *loop()*, která je určena pro opakující se kód. Dále jsou v souboru main.cpp propojeny všechny výše zmíněné komponenty.

Ve funkci *setup()* je nejprve předána tzv. callback funkce komponentě *Bluetooth*, skrze metodu *setBaseStationGeometryCB()*. Touto callback metodou je poté předána nová geometrie, která byla přijata přes Bluetooth. Poté jsou zavolány metody *setup()* nad instancemi *Bluetooth* a *LedDriver*, pomocí kterých jsou provedeny potřebné inicializace v daných instancích. Následuje konfigurace periferií MCU tak, aby bylo možné měřit přesně délky pulzů, které se objevují na výstupním pinu *D* integrovaného obvodu TS4231. Jedná se o periferie GPIOTE, PPI a TIMER. Periferie GPIOTE (GPIO Task and Events) umožňuje skrze registr CONFIG nakonfigurovat až 8 kanálů, které mohou být využity pro zachytávání změny logické úrovně na GPIO (General Purpose Input Output) pinech. Protože je potřeba měřit délky zachycených pulzů, je nutné zachytávat změnu logické úrovně z nízké na vysokou a zároveň i z vysoké úrovně na nízkou na GPIO pinu MCU, ke kterému je

připojený pin *D* integrovaného obvodu TS4231. Nejprve jsou tedy nakonfigurovány dva kanály periferie GPIOTE, přičemž nultý kanál zachytává nástupnou hranu a první kanál hranu sestupnou. Poté musí být povoleny přerušení generované periferií GPIOTE při změně logické úrovně na pinu *D* pro oba kanály skrze registr INTENSET. Dále musí být rovněž povoleno externí přerušení pro periferii GPIOTE zavoláním funkce *NVIC_EnableIRQ()*, kde v parametru funkce je předáno příslušné číslo přerušení. Je to z důvodu, aby procesor mohl zareagovat na událost o změně hrany na pinu *D* a zkopírovat tak obsah registru CC do RAM paměti obsahující příslušné časy. Následuje konfigurace periferie TIMER, pomocí které lze přesně měřit délku pulzů, tj. dobu mezi nástupnou a sestupnou hranou na pinu *D*. V nRF52840 je k dispozici až 5 periferií TIMER, avšak v případě využívání Bluetooth je SoftDevicem vyžadován TIMER0. Jelikož Bluetooth je využíváno, byl pro měření délky pulzů zvolen TIMER1. TIMER1 je skrze registr MODE nastaven do režimu časovače. Dále je nastavena předdělička základní frekvence 16 MHz na hodnotu 0, tj. předdělička nebude využívána. Je to z důvodu, kdy měření délky pulzů a doby mezi nástupnými hranami přímo ovlivňuje přesnost trackování pozice. Takto bude délka jednoho ticku TIMER1 trvat 62,5 ns. Poté je nastavena maximální hodnota TIMER1 skrze registr BITMODE, přičemž byla nastavena maximální možná hodnota o délce 32 bitů. Periferie TIMER umožňuje zkopírování aktuální hodnoty registru COUNTER do CC registru po zapsání hodnoty do registru TASKS_CAPTURE. Jakmile byly nakonfigurovány periferie GPIOTE a TIMER, je potřeba tyto periferie propojit. Na platformě nRF52840 existuje periferie PPI (Programmable Peripheral Interconnect), která umožňuje komunikaci mezi periferiemi bez nutnosti zásahu CPU. Tato periferie nabízí až 32 kanálů, přičemž prvních 20 může být libovolně využito. Pro přenášení událostí z periferie GPIOTE do periferie TIMER byly využity první dva kanály. Nultý kanál byl opět použit pro přenášení události o nástupné hraně do TIMER1. Registr EEP periferie PPI umožňuje nastavit vstupní událost, která je přenášena skrze periferii PPI. Zde do registru EEP byla přiřazena událost vyvolaná nástupnou hranou na pinu *D* integrovaného obvodu TS4231. Potom je skrze registr TEP nastavena úloha, která má být vykonána v případě, že nastane událost nakonfigurovaná skrze registr EEP. Zde je nastavena úloha TASKS_CAPTURE periferie TIMER1, přičemž offset adresy TASKS_CAPTURE určuje,

do kterého registru CC je zapsána aktuální hodnota TIMER1. V případě nástupné hrany je opět offset nastaven na hodnotu 0. Poté následuje stejný postup konfigurace periferie PPI i pro sestupnou hranu, kde je rozdíl pouze v použitých kanálech neboli offsetech. Potom je ještě nutné povolit oba kanály pro přenos událostí o změně hrany skrze registr CHENSET. V tento moment jsou nakonfigurovány všechny potřebné periferie – GPIOTE, TIMER a PPI. Zbývá tak tedy již jen spustit TIMER1 skrze zapsání hodnoty 1 do registru TASKS_START periferie TIMER1. Celou konfiguraci těchto tří periferií ve funkci *setup()* je možné vidět v ukázce kódu 1. Nakonec se ještě ve funkci *setup()* čeká maximálně 500 ms, zdali byl 3D trackerem zachycen nějaký IR paprsek. V případě, že žádný IR paprsek zachycen není, je komponentě *LedDriver* vydán pokyn, aby přes LED diodu informoval uživatele o této skutečnosti.

```
// setting GPIOTE channels for raising and falling edge
uint8_t pin_D = PIN_D_TS4231.number;
uint8_t port_pin_D = PIN_D_TS4231.port;
uint8_t gpiote_channel_raising = 0;
uint8_t gpiote_channel_falling = 1;

NRF_GPIOTE->CONFIG[gpiote_channel_raising] =
  ((uint32_t) GPIOTE_CONFIG_MODE_Event << GPIOTE_CONFIG_MODE_Pos)
  | ((uint32_t) pin_D << GPIOTE_CONFIG_PSEL_Pos)
  | ((uint32_t) port_pin_D << 13UL)
  | ((uint32_t) GPIOTE_CONFIG_POLARITY_LoToHi << GPIOTE_CONFIG_POLARITY_Pos);
NRF_GPIOTE->CONFIG[gpiote_channel_falling] =
  ((uint32_t) GPIOTE_CONFIG_MODE_Event << GPIOTE_CONFIG_MODE_Pos)
  | ((uint32_t) pin_D << GPIOTE_CONFIG_PSEL_Pos)
  | ((uint32_t) port_pin_D << 13UL
  | ((uint32_t) GPIOTE_CONFIG_POLARITY_HiToLo << GPIOTE_CONFIG_POLARITY_Pos);
NRF_GPIOTE->INTENSET =
  ((uint32_t) GPIOTE_INTENSET_IN0_Enabled << GPIOTE_INTENSET_IN0_Pos)
  | ((uint32_t) GPIOTE_INTENSET_IN1_Enabled << GPIOTE_INTENSET_IN1_Pos);

NVIC_EnableIRQ(GPIOTE_IRQn);

// config TIMER1 for capturing timestamps of rising and falling edges from GPIOTE
NRF_TIMER1->MODE = TIMER_MODE_MODE_Timer;
NRF_TIMER1->PRESCALER = 0;    // 16MHz
NRF_TIMER1->BITMODE = TIMER_BITMODE_BITMODE_32Bit;

// config PPI for transfering events of rising/falling edge on pin D to TIMER1
for capturing timestamps
NRF_PPI->CH[gpiote_channel_raising].EEP =
  (uint32_t) &NRF_GPIOTE->EVENTS_IN[gpiote_channel_raising];
NRF_PPI->CH[gpiote_channel_raising].TEP =
  (uint32_t) &NRF_TIMER1->TASKS_CAPTURE[gpiote_channel_raising];
NRF_PPI->CH[gpiote_channel_falling].EEP =
  (uint32_t) &NRF_GPIOTE->EVENTS_IN[gpiote_channel_falling];
NRF_PPI->CH[gpiote_channel_falling].TEP =
  (uint32_t) &NRF_TIMER1->TASKS_CAPTURE[gpiote_channel_falling];
NRF_PPI->CHENSET =
  ((uint32_t) PPI_CHENSET_CH0_Enabled << PPI_CHENSET_CH0_Pos)
  | ((uint32_t) PPI_CHENSET_CH1_Enabled << PPI_CHENSET_CH1_Pos);

// start TIMER1
NRF_TIMER1->TASKS_START = 1;
```

Ukázka kódu č. 1 Konfigurace periferií pro zachytávání délek pulzů na pinu *D* ve funkci *setup()*.

Ve funkci *loop()* se volají loop metody nad třemi komponentami, které potřebují pravidelně zpracovávat data. Jedná se o komponenty *PulseProcessor*, *Bluetooth* a *LedDriver*. Na konci funkce *loop()* se testuje, zdali nebyl 3D tracker zastíněn před Lighthouse stanicemi. Pokud byl, je nad instancí *LedDriver* zavolána metoda *showTrackingTimeout()*, která uživateli pomocí LED zobrazí tuto informaci.

V souboru main.cpp je přítomna ještě obsluha přerušení periferie GPIOTE a dvě callback funkce. První callback funkce *baseStationGeometryCB()* přijímá v parametru pointer na strukturu *BaseStation*, skrze který se předává pointer na dvouprvkové pole obsahující geometrie Lighthouse stanic. Uvnitř funkce *baseStationGeometryCB()* jsou poté přijaté geometrie nastaveny komponentě *PositionCalculator* a je nastaven stav 3D trackeru na přijímání souřadnic. Callback funkci *baseStationGeometryCB()* je možné vidět v ukázce kódu 2.

```
void baseStationGeometryCB(BaseStation *baseStations) {
  positionCalculator.setBaseStationsGeometry(
    baseStations[0], baseStations[1]);
  // clear state showing that geometry is not set
  receivingCoordinates = true;
  ledDriver.showTracking();
}
```

Ukázka kódu č. 2 Callback funkce skrze kterou je předávána nová geometrie přijatá přes Bluetooth komponentě *PositionCalculator*.

Callback funkce *anglesCB()* je předána komponentě *PulseProcessor* při vytváření instance v parametru konstruktoru. Touto callback metodou *PulseProcessor* předává poiner na floatové pole délky 4, ve kterém jsou uloženy 4 zachycené úhly z obou Base Station. Ve funkci *anglesCB()* je poté iniciován výpočet pozice 3D trackeru zavoláním metody *calcPosition()* nad instancí *PositionCalculator*. Tato metoda přijímá v parametrech pointer na 4 úhly ze dvou Base Station a pointer na strukturu *vec3d*, do které jsou v případě úspěšného výpočtu uloženy souřadnice *x*, *y* a *z* 3D trackeru v prostoru. Metoda *calcPosition()* vrací celočíselnou hodnotu vyjadřující výsledek provedeného výpočtu. V případě, kdy byla pozice v pořádku vypočtena jsou souřadnice 3D trackeru odeslány skrze Bluetooh a je nastaven příslušný stav 3D trackeru. V opačném případě je změněn stav 3D trackeru tak, aby

byl uživatel informován o daném problému. Celá callback funkce *anglesCB()* je uvedena v ukázce kódu 3.

```
void anglesCb(float32_t *angles) {
    vec3d xyz;
    uint8_t result = positionCalculator.calcPosition(angles, xyz);

    switch(result) {
        case CALCULATION_SUCCESS:
            lastPositionUpdateMillis = millis();
            bluetooth->sendPosition(xyz);
            ledDriver.showTracking();
            receivingCoordinates = true;
            break;
        case BASESTATIONS_NOT_SET:
            ledDriver.showGeometryNotSet();
            receivingCoordinates = false;
            break;
        case CALCULATION_FAILED:
            ledDriver.showFailed();
            receivingCoordinates = false;
            break;
    }
}
```

Ukázka kódu č. 3 Callback funkce skrze kterou jsou předány 4 úhly z komponenty *PulseProcessor* do komponenty *PositionCalculator*.

Funkce obsluhující GPIOTE přerušení musí být pojmenována jako *GPIOTE_IRQHandler*. Tato funkce je volána s každým přerušením, které je generované periferií GPIOTE. Jsou tedy zde uvnitř dvě podmínky, které testují, zdali je na nultém nebo prvním kanálu GPIOTE příznak v registru EVENTS_IN. Nultý kanál odpovídá nástupné hraně, zatímco kanál první sestupné hraně. Jakmile je zachyceno jedno z přerušení, je nutné vždy tento příznakový bit vynulovat. V případě detekování nástupné hrany je nastavena booleovská proměnná *lightDetected* na hodnotu *true*, přičemž na základě hodnoty *lightDetected* je na začátku programu rozsvícen stav indikující, že žádný IR paprsek nebyl detekován. Praktičtější je však detekování sestupné hrany, neboť pro získání délky pulzu je vždy nutné vědět čas začátku a čas konce pulzu. V podmínce testující sestupnou hranu je tak nejprve zkopírován obsah registru CC do paměti RAM. Poté je vypočten rozdíl časů mezi sestupnou a nástupnou hranou, vytvořen pulse definovaný strukturou *Pulse* a

nakonec je tento pulse předán komponentně *PulseProcessor*, která jej později zpracuje v *loop* smyčce. Výsledný kód obsluhy přerušení periferie GPIOTE je uveden v ukázce kódu 4.

```
extern "C" {  //important; otherwise irq handler won't be called
    void GPIOTE_IRQHandler(void) {
        if(NRF_GPIOTE->EVENTS_IN[0] != 0) {
            NRF_GPIOTE->EVENTS_IN[0] = 0;
            // raising edge
            lightDetected = true;
        }

        if(NRF_GPIOTE->EVENTS_IN[1] != 0) {
            NRF_GPIOTE->EVENTS_IN[1] = 0;
            // falling edge
            uint32_t raising_time = NRF_TIMER1->CC[0];
            uint32_t falling_time = NRF_TIMER1->CC[1];
            uint32_t pulse_length_ticks = falling_time - raising_time;

            Pulse pulse = {raising_time, pulse_length_ticks};
            pulseProcessor.addPulse(pulse);
        }
    }
}
```

Ukázka kódu č. 4 Obsluha přerušení periferie GPIOTE.

6.1.2 Komponenta PulseProcessor

PulseProcessor je zodpovědný za zpracovávání zachycených pulzů. Uvnitř komponenty *PulseProcessor* je využíván kruhový buffer, do kterého se přidávají nově zachycené pulzy skrze funkci *addPulse()*, která v parametru přijímá strukturu *Pulse*. Pulzy jsou přidávány s každou zachycenou sestupnou hranou uvnitř obsluhy GPIOTE přerušení. Délka kruhového bufferu je 10. Metoda *addPulse()* objektu *PulseProcessor* je uvedena v ukázce kódu 5.

```
void PulseProcessor::addPulse(Pulse pulse) {
    uint8_t w = (pulsesWriteIndex + 1) % PULSES_COUNT;
    uint8_t r = pulsesReadIndex % PULSES_COUNT;
    if(w == r) {
        // the data would be overwritten, let's reset cycle
        resetCycle();
        // reset read and write indexes too
        pulsesReadIndex = 0;
        pulsesWriteIndex = 0;
    }
    pulses[pulsesWriteIndex++ % PULSES_COUNT] = pulse;
}
```

Ukázka kódu č. 5 Metoda *addPulse()* třídy *PulseProcessor* odpovědná za přidávání pulzů do kruhového bufferu.

Pulzy přidané do kruhového bufferu jsou poté zpracovávány CPU vždy v *loop* smyčce. Za tímto účelem třída *PulseProcessor* obsahuje metodu *loop()*. Nejprve je ve funkci *loop()* přečten pulz, který je aktuálně na řadě. Pokud je buffer s pulzy prázdný, je vykonávání funkce *loop()* přerušeno. Následuje dekódování 3 bitů z délky pulzu zavoláním metody *decodePulse()*. Metoda vrací celočíselnou hodnotu, vyjadřující 3 bity informace z délky pulzu. Dekódovaní 3 bitů bylo provedeno podle výpočtu uvedeného v [11], avšak délky pulzů musely být upraveny. Autorem diplomové práce byly délky pulzů zachycených pomocí 3D trackeru posunuté cca o 3 µs nahoru oproti ideálním délkám pulzů. Byly proto stanoveny nové délky pulzů posunuté cca o 3 µs nahoru. Nově sestavené intervaly délek pulzů jsou uvedeny v tabulce 5.

Tabulka 5 Nově stanovené délky pulzů posunuté cca o 3 µs nahoru

Skip	Data	Axis	Interval [ticks]		Délka [µs]
0	0	0	889	1053	65,8
0	0	1	1054	1218	76,1
0	1	0	1219	1383	86,4
0	1	1	1384	1548	96,8
1	0	0	1549	1713	107,1
1	0	1	1714	1878	117,4
1	1	0	1879	2043	127,7
1	1	1	2044	2208	138

Přepočet délky pulzu v tickách na hodnotu obsahující 3 bity informací je proveden jako

$$[skip, data, axis] = (length - 889) \,/\, 165. \tag{23}$$

Jakmile jsou získány bity jako celé číslo, je nejprve zkontrolováno, zdali není číslo mimo rozsah 0 – 7. Pokud je hodnota proměnné *bits* záporná, znamená to, že se jedná o krátký pulz a mělo by se tak jednat o zásah laserem. V případě, že je cyklus synchronizován, je proveden přepočet podle vzorce 3 v metodě *computeAngle()*. Pokud je však hodnota proměnné *bits* větší než 7, jedná se o neplatný pulz a cyklus je tak resetován skrze metodu *resetCycle()*. V případě, že se jedná o validní synchronizační pulz, jsou extrahovány 3 bity informací pomocí bitových operací posun doprava a AND. Výše zmíněný začátek metody *loop()* je možné vidět v ukázce kódu 6.

```
void PulseProcessor::loop(void) {
    Pulse pulse;
    if(!readPulse(pulse)) return;   // nothing to do

    int8_t bits = decodePulse(pulse);

    if(bits < 0) {
        // sweep pulse
        if(!cycleSynchronized) return;
        computeAngle(pulse);

        return;
    }

    if(bits > 7) {
        // invalid length of pulse
        resetCycle();

        return;
    }

    // extract bits
    uint8_t skip = ((uint8_t) bits >> 2) & 1;
    uint8_t data = ((uint8_t) bits >> 1) & 1;
    uint8_t axis = ((uint8_t) bits) & 1;
```

Ukázka kódu č. 6 Začátek metody *loop()*.

Následuje blok kódu v podmínce pro hledání začátku celého cyklu sestávající se ze 4 cyklů. Pro začátek cyklu je typické, že první synchronizační pulz patří master Base Station, přičemž master Base Station aktuálně vrhá laserový paprsek do místnosti v horizontální ose, tj. hodnota bitů axis a skip je 0. Následně musí přijít synchronizační pulz slave Base Station. Tento pulz musí přijít zhruba o 400 µs později. Pokud jsou tyto podmínky splněny, je nalezen začátek celého cyklu. Kód provádějící hledání začátku cyklu v metodě *loop()* je možné vidět v ukázce kódu 7.

```
// find start of full cycle - situation where master base station sweeps room
in axis 0
if(!cycleSynchronized) {
    if(!skip && !axis) {
        // candidate for master's lighthouse sync pulse in axis 0 (start of
        cycle)
        startTime = pulse.raising_time_ticks;
        startCycleCandidate = true;
        baseStationIndex = 1;

        return;
    }
    if(startCycleCandidate) {
        // candidate pulse was detected
        uint32_t diff = pulse.raising_time_ticks - startTime;
        if(diff >= MICROSECS_IN_TICKS(390) &&
                diff <= MICROSECS_IN_TICKS(430)) {
            // start of cycle was detected
            cycleSynchronized = true;
            baseStationIndex = 2;

            return;
        } else {
            // start was not detected, reset candidate
            startCycleCandidate = false;
        }
    }
}
if(!cycleSynchronized) return;
```

Ukázka kódu č. 7 Hledání začátku celého cyklu v metodě *loop()*.

Jakmile je nalezen začátek cyklu, následuje zpracovávání synchronizačních pulzů. Zde se využívá skutečnosti, že celkový cyklus pro získání 4 úhlů tvoří 4 cykly, pro které je typické, že nejprve přijde synchronizační pulz master Base Station, poté o zhruba 400 µs později synchronizační pulz slave Base Station a potom případně krátký pulz laseru. Využívá se tak dvou indexů – *baseStationIndex* a *cycle*, kdy první tvoří číslo cyklu a druhý číslo aktuální Base Station. Nejprve je testován aktuální index Base Station. Pokud patří aktuální synchronizační pulz slave Base Station, je otestováno, zdali dorazil tento pulz v přípustném časovém intervalu. Pokud ne, je cyklus resetován a hledá se tak znovu úplný začátek cyklu. Jestliže je hodnota proměnné *baseStationIndex* rovna 2, znamená to, že nastal konec jednoho cyklu,

proměnná *cycle* je inkrementována o jedničku a proměnná *baseStationIndex* je vynulována. Výše zmíněný kód je možné vidět v ukázce kódu 8.

```
switch(baseStationIndex) {
    case 1:
        {
            uint32_t diff = pulse.raising_time_ticks - startOfCycleTime;
            if(!(diff >= MICROSECS_IN_TICKS(390) &&
                        diff <= MICROSECS_IN_TICKS(430))) {
                // some sync pulses were missed
                resetCycle();

                return;
            }
        }
        break;
    case 2:     // end of one cycle
        cycle++;
        baseStationIndex = 0;
        break;
}
```

Ukázka kódu č. 8 Kontrola indexu Base Station v metodě *loop()*.

Dále se hledá, zdali nenastal konec celého cyklu tvořící 4 cykly. Pokud ano a byly vypočteny všechny 4 úhly, je zavolána callback funkce *anglesCB()* v souboru main.cpp, skrze kterou jsou předány zachycené úhly. Potom jsou vyresetovány proměnné *cycle* a *anglesCount*. Poté je do proměnné *startTime* uložen čas náběžné hrany správného synchronizačního pulzu Base Station, od kterého je poté vypočítán úhel. V prvních dvou cyklech se určují úhly master Base Station, zatímco ve třetím a čtvrtém cyklu se určuje úhel pro slave Base Station. Následuje hledání začátku jednoho cyklu. V případě, že je začátek nalezen, tj. hodnota proměnné *baseStationIndex* je rovna 0, je uložen začátek do proměnné *startOfCycleTime*. Nakonec metody *loop()* je ještě inkrementována proměnná *baseStationIndex* o jedničku. Právě popsaný kód je možné vidět v ukázce 9.

```
    if(cycle == CYCLES_COUNT) { // end of full cycle (4 cycles = 4 angles)
        if(anglesCount == ANGLES_COUNT) {
            angleCb(angles);   // let's propagate caught angles
        }
        cycle = 0;
        anglesCount = 0;
    }

    if((cycle / 2 == 0 && baseStationIndex == 0) ||
        (cycle / 2 == 1 && baseStationIndex == 1)) {
        startTime = pulse.raising_time_ticks;
    }

    if(baseStationIndex == 0) { // start of one cycle
        startOfCycleTime = pulse.raising_time_ticks;
    }
    baseStationIndex++;
}
```

Ukázka kódu č. 9 Konec metody *loop()*, propagování úhlů, hledání časových začátků.

Metoda *computeAngle()* provádí přepočet z pulzů na úhel. Tato metoda přebírá v parametru pointer na strukturu *Pulse*. Nejprve je vypočtena doba v tickách jako rozdíl mezi středem délky laserového pulzu a časem nástupné hrany příslušného synchronizačního pulzu. Poté následují dvě validace. První validace testuje, zdali vypočtený čas je ve validním časovém rozmezí pro úhel. Pokud ano, je ještě otestováno, zdali pro aktuální cyklus nebyl již úhel vypočten. To může být způsobeno odrazy laseru v prostoru. V případě, že úhel přepočten ještě nebyl, je proveden výpočet úhlu podle vzorce 3 a inkrementován aktuální počet úhlů o jedničku. Celou implementaci metody *computeAngle()* je možné vidět v ukázce kódu 10.

```
void PulseProcessor::computeAngle(Pulse &pulse) {
    uint32_t time = (pulse.raising_time_ticks + pulse.length_ticks / 2)
                          - startTime;
    if(time < sweepStartWindowInTicks || time > sweepEndWindowsInTicks) {
        // sweep pulse was in invalid range
        return;
    }

    if(anglesCount > cycle) {
        // second "sweep" pulse in one cycle, could be some kind of reflection
        return;
    }

    angles[cycle] = ((int32_t) time - (int32_t) centerCycleLengthInTicks) * PI
                          / (int32_t) cyclePeriodLengthInTicks;
    anglesCount++;
}
```

Ukázka kódu č. 10 Metoda *computeAngle()* odpovědná za přepočet času na úhel.

6.1.3 Komponenta PositionCalculator

Třída *PositionCalculator* slouží pro výpočet aktuální pozice 3D trackeru v prostoru z aktuálních úhlů mezi Lighthouse stanicemi a 3D trackerem. Pro výpočet pozice je ve třídě metoda *calcPosition()*. Aby mohl být proveden výpočet, musí být nastavena geometrie Lighthouse stanic. Proto je v metodě *calcPosition()* vždy nejprve zkontrolováno, zdali byla geometrie nastavena. Pokud tomu tak není, je vrácena celočíselná hodnota vyjadřující, že geometrie nastavená není. Potom jsou vypočteny dva vektory směřující od Lighthouse stanic k fotodiodě na 3D trackeru zavoláním metody *calcRayVector()*. Jakmile jsou známy vektory, je vypočten jejich průsečík zavoláním metody *intersectLines()*. Kód metody *calcPosition()* je možné vidět v ukázce kódu 11.

```
int PositionCalculator::calcPosition(float32_t *angles, vec3d &position) {
    // check whether base stations geometry is set
    for(uint8_t b = 0; b < baseStationsCount; b++) {
        if(!baseStations[b].isSet()) {
            return BASESTATIONS_NOT_SET;
        }
    }

    vec3d ray1, ray2;
    calcRayVector(baseStations[0], angles[0], angles[1], ray1);
    calcRayVector(baseStations[1], angles[2], angles[3], ray2);

    return intersectLines(baseStations[0].origin, ray1,
                                baseStations[1].origin, ray2, position);
}
```

Ukázka kódu č. 11 Metoda *calcPosition()* třídy *PositionCalculator* sloužící pro výpočet pozice 3D trackeru.

Metoda *calcRayVector()* slouží, jak již bylo zmíněno, pro výpočet vektoru mezi Base Station a fotodiodou 3D trackeru. Metoda má 4 vstupní parametry. První parametr obsahuje pointer na geometrii Base Station, druhý parametr je horizontální úhel, třetí parametr je vertikální úhel a poslední parametr je pointer na strukturu *vec3d*, do které bude uložen výsledný vektor. V metodě jsou nejprve vypočteny souřadnice normálových vektorů rovin podle vzorců 5, 6. Poté je vypočten vektor mezi Base Station a fotodiodou 3D trackeru jako vektorový součin vektorů *vertical* a *horizontal* podle vzorce 7. Následuje převod vypočteného vektoru z lokálního souřadného systému Base Station do globálního vynásobením vektoru rotační maticí Base Station. Výsledný vektor v globálním souřadném systému je poté uložen na adresu v paměti do proměnné *res*. Výsledný kód metody *calcRayVector()* je možné vidět v ukázce kódu 12.

```
void PositionCalculator::calcRayVector(BaseStation &baseStation,
       float32_t angle1, float32_t angle2, vec3d &res) {
    vec3d vertical = {arm_cos_f32(angle1), 0, -arm_sin_f32(angle1)};
    vec3d horizontal = {0, arm_cos_f32(angle2), arm_sin_f32(angle2)};

    vec3d ray = {};
    vectorCrossProduct(horizontal, vertical, ray);
    normalizeVector(ray);

    arm_matrix_instance_f32 stationRotMatrix = {3, 3,baseStation.rotationMatrix};
    arm_matrix_instance_f32 rayVector = {3, 1, ray};
    arm_matrix_instance_f32 rotatedRayVector = {3, 1, res};
    arm_mat_mult_f32(&stationRotMatrix, &rayVector, &rotatedRayVector);
}
```

Ukázka kódu č. 12 Metoda *calcRayVector()* počítající vektor mezi Base Station a fotodiodou 3D trackeru.

Za výpočet průsečíku dvou vektorů od Lighthouse stanic je zodpovědná metoda *intersectLines()*. Metoda *intersectLines()* přijímá 5 parametrů, přičemž první parametr je pozice první Base Station, druhý parametr je vektor od první Base Station, třetím parametrem je pozice druhé Base Station, čtvrtý parametr je vektor od druhé Base Station a poslední pátý parametr je pointer na strukturu *vec3d*, do které je uložena výsledná pozice. Nejprve je nalezena vzdálenost *w0* mezi středy Lighthouse stanic podle vzorce 12. Následuje výpočet pěti skalárních součinů vektorů podle vzorců 13 – 17. Poté je vypočten jmenovatel ze vzorců 18, 19. Dále je otestována hodnota vypočteného jmenovatele, aby jeho hodnota nebyla příliš blízká 0, neboť by vektory od Lighthouse stanic mohly být rovnoběžné. Pokud by nastala tato situace, je výpočet ukončen a funkcí *intersectLines()* je vrácena celočíselná hodnota vyjadřující tuto skutečnost. Následně jsou vypočteny parametry *s*, *t* podle vzorců 18, 19 a také body na vektorech s nejkratší vzdáleností mezi sebou podle vzorců 20, 21. Nakonec je vypočten střední bod mezi vypočtenými body *ps*, *pt* podle vzorce 22, jenž odpovídá pozici 3D trackeru v prostoru. Na závěr je funkcí *intersectLines()* vrácena celočíselná hodnota vyjadřující úspěšný výpočet průsečíků dvou vektorů. Implementaci metody *intersectLines()* je možné vidět v ukázce kódu 13.

```
int PositionCalculator::intersectLines(vec3d &origin1, vec3d &vec1,
       vec3d &origin2, vec3d &vec2, vec3d &res) {
    vec3d w0 = {};
    arm_sub_f32(origin1, origin2, w0, vec3dSize);

    float32_t a, b, c, d, e;
    arm_dot_prod_f32(vec1, vec1, vec3dSize, &a);
    arm_dot_prod_f32(vec1, vec2, vec3dSize, &b);
    arm_dot_prod_f32(vec2, vec2, vec3dSize, &c);
    arm_dot_prod_f32(vec1, w0, vec3dSize, &d);
    arm_dot_prod_f32(vec2, w0, vec3dSize, &e);

    float32_t denominator = a * c - b * b;
    if(fabsf(denominator) < 1e-5f)
        return CALCULATION_FAILED;

    float32_t s = (b * e - c * d) / denominator;
    vec3d ps = {};
    arm_scale_f32(vec1, s, ps, vec3dSize);
    arm_add_f32(ps, origin1, ps, vec3dSize);

    float32_t t = (a * e - b * d) / denominator;
    vec3d pt = {};
    arm_scale_f32(vec2, t, pt, vec3dSize);
    arm_add_f32(pt, origin2, pt, vec3dSize);

    vec3d aux = {};
    arm_add_f32(ps, pt, aux, vec3dSize);
    arm_scale_f32(aux, 0.5f, res, vec3dSize);

    return CALCULATION_SUCCESS;
}
```

Ukázka kódu č. 13 Metoda *intersectLines()* sloužící pro výpočet průsečíku mezi dvěma různoběžnými vektory.

6.1.4 Komponenta Bluetooth

Třída *Bluetooth* slouží pro komunikaci skrze Bluetooth. Práce s Bluetooth je jednoduchá, jelikož bylo využito frameworku pro Arduino. V této třídě jsou vytvořeny instance BLE služeb a charakteristik dle tabulky 4. Dále tato třída spravuje geometrii přijatou skrze BLE, přičemž pokud byla přijata geometrie, je tato geometrie propagována dále do souboru main.cpp skrze callback funkci

baseStationGeometryCB(). V neposlední řadě umožňuje třída *Bluetooth* odesílání aktuální pozice 3D trackeru skrze mechanismus notify charakteristiky Coordinates.

Ze třídy *Bluetooth* budou popsány následující metody: *setup()*, *poll()*, *sendPosition()*, *stationNumberWritten()*, *positionWritten()* a *rotationWritten()*.

Metoda *setup()* je volána při inicializaci programu. V této funkci je nejprve provedena počáteční inicializace geometrie Lighthouse stanic vynulováním všech prvků vektoru a rotační matice. Následuje nastavení peripheral role. Nejprve je nastaveno jméno zařízení, následně je nastaveno UUID služby Coordinates, které je odesíláno v advertise paketu. Na základě UUID této služby jsou poté 3D trackery filtrovány v mobilní aplikaci. Poté jsou všechny služby a charakteristiky přidány do BLE vrstvy GATT skrze metodu *addAttribute()*. Dále jsou pro každou charakteristiku přiřazeny callback metody pro obsluhu události při operaci zápis. Nakonec je v této metodě spuštěna role peripheral zavoláním metody *begin()*. Implementaci metody *setup()* je možné vidět v ukázce kódu 14.

```
void Bluetooth::setup(void) {
    // at the start zeros base station's values
    for(uint8_t b = 0; b < baseStationsCount; b++) {
        memset(baseStations[b].origin, 0, sizeof(vec3d));
        memset(baseStations[b].rotationMatrix, 0, sizeof(mat3Array));
    }

    peripheral.setLocalName("3D Tracker");
    peripheral.setAdvertisedServiceUuid(coordinatesService.uuid());

    peripheral.addAttribute(coordinatesService);
    peripheral.addAttribute(coordinatesCharacteristic);

    peripheral.addAttribute(baseStationGeometryService);
    peripheral.addAttribute(stationNumberCharacteristic);
    peripheral.addAttribute(positionCharacteristic);
    peripheral.addAttribute(rotation1RowCharacteristic);
    peripheral.addAttribute(rotation2RowCharacteristic);
    peripheral.addAttribute(rotation3RowCharacteristic);

    stationNumberCharacteristic.setEventHandler(BLEWritten,
                                                charStationNumberWritten);
    positionCharacteristic.setEventHandler(BLEWritten, charPositionWritten);
    rotation1RowCharacteristic.setEventHandler(BLEWritten, charRotationWritten);
    rotation2RowCharacteristic.setEventHandler(BLEWritten, charRotationWritten);
    rotation3RowCharacteristic.setEventHandler(BLEWritten, charRotationWritten);

    peripheral.begin();
}
```

Ukázka kódu č. 14 Implementace metody *setup()* ve třídě *Bluetooth*.

Následuje popis velmi důležité metody *poll()*. V této metodě je zavolána metoda *poll()* nad instancí *BLEPeripheral*. Tato metoda musí být volána neustále ve smyčce *loop*, aby bylo obsluhováno Bluetooth. Pokud by došlo k vynechání tohoto volání, Bluetooth by nebylo funkční.

Skrze metodu *sendPosition()* je odesílána nová pozice 3D trackeru, která je předávána v parametru metody. Uvnitř metody je nejprve převeden datový typ vektoru na interní datový typ využívaný Bluetooth charakteristikou. Následně je nastavena hodnota charakteristiky Coordinates skrze metodu *setValue()*. Protože byla charakteristice Coordinates nastavena vlastnost notify, je po zavolání metody

setValue() automaticky odeslána tato hodnota do smartphonu, který o to požádal. Implementaci metody *sendPosition()* je možné vidět v ukázce kódu 15.

```
void Bluetooth::sendPosition(vec3d &position) {
    Vec3 coords = {position[0], position[1], position[2]};
    coordinatesCharacteristic.setValue(coords);
}
```

Ukázka kódu č. 15 Implementace metody *sendPosition()* ve třídě *Bluetooth*.

Metoda *stationNumberWritten()* je callback metoda, která je volána vždy, když je do charakteristiky Station Number zapsána central zařízením jakákoliv hodnota. Tato charakteristika nastavuje index geometrie Base Station v případě, že zapsaná hodnota je validní. Pokud je index validní, musí být nastaveny hodnoty charakteristik Position a všech Rotation Row. Jestliže nebyl index validní, je charakteristice Station Number nastavena původní hodnota před zápisem. Implementace metody *stationNumberWritten()* je uvedena v ukázce kódu 16.

```
void Bluetooth::stationNumberWritten(BLECentral &central,
       BLECharacteristic &characteristic) {
    uint8_t value = characteristic.value()[0];

    if(value < baseStationsCount &&
       characteristic.valueLength() == sizeof(uint8_t)) {
        // set the new written base station index
        baseStationIndex = value;

        // update characteristic values
        setPositionCharValueByIndex();
        setRotationCharValueByIndex();
    } else {
        // restore the previous base station index
        stationNumberCharacteristic.setValue(baseStationIndex);
    }
}
```

Ukázka kódu č. 16 Implementace metody *stationNumberWritten()* ze třídy *Bluetooth*.

Metoda *positionWritten()* je také callback metoda, zde ale pro zápis pozice Base Station. Nejprve je opět zkontrolováno, zdali jsou zapsána validní data. Pokud jdou data validní, tak je nastavena pozice Base Station podle hodnoty proměnné *baseStationIndex*. V případě, že zapsaná data validní nebyla, je opět obnovena

původní hodnota charakteristiky Position služby Base Station Geometry. Výsledná implementace metody *positionWritten()* je uvedena v ukázce kódu 17.

```
void Bluetooth::positionWritten(BLECentral &central,
       BLECharacteristic &characteristic) {
    if(characteristic.valueLength() == sizeof(Vec3)) {
        // set the new written position to the base station
        Vec3 vec = positionCharacteristic.value();
        memcpy(baseStations[baseStationIndex].origin, &vec, sizeof(Vec3));
    } else {
        // restore the previous position from base station
        setPositionCharValueByIndex();
    }
}
```

Ukázka kódu č. 17 Implementace metody *positionWritten()*.

Poslední popisovaná metoda *rotationWritten()* se využívá rovnou pro tři charakteristiky. Nejprve je zkontrolována délka zapsaných dat v charakteristice. Pokud délka zapsaných dat neodpovídá délce v charakteristikách Rotation Row, jsou obnovena data všech charakteristik Rotation Row. Následuje porovnání UUID charakteristik pomocí funkce *memcmp()*. Následně podle výsledku funkce *memcmp()* je nalezena charakteristika Rotation Row, do které byla zapsána data. Poté je přijatý řádek nastaven rotační matici aktuální geometrie Base Station podle hodnoty proměnné *baseStationIndex*. Nakonec se otestuje, zda je nastavena kompletní geometrie obou Base Station a to tak, když jsou data geometrie porovnávána, aby se lišila od úplné nuly. Pokud tomu tak je, je výsledná geometrie obou Base Station předána zavoláním callback funkce *baseStationGeometryCB()* v souboru main.cpp. Implementaci metody *rotationWritten()* je možné vidět v ukázce kódu 18.

```
void Bluetooth::rotationWritten(BLECentral &central,
       BLECharacteristic &characteristic) {
    if(characteristic.valueLength() != sizeof(Vec3)) {
        // restore the previous rotation from base station
        setRotationCharValueByIndex();

        return;
    }

    int rot1Row =
       memcmp(characteristic.uuid(), rotation1RowCharacteristic.uuid(), 16);
    int rot2Row =
       memcmp(characteristic.uuid(), rotation2RowCharacteristic.uuid(), 16);
    int rot3Row =
       memcmp(characteristic.uuid(), rotation3RowCharacteristic.uuid(), 16);

    if(rot1Row == 0) {
        Vec3 rot1 = rotation1RowCharacteristic.value();
        setRowToRotationMatrix(0, rot1);
    } else if(rot2Row == 0) {
        Vec3 rot2 = rotation2RowCharacteristic.value();
        setRowToRotationMatrix(1, rot2);
    } else if(rot3Row == 0) {
        Vec3 rot3 = rotation3RowCharacteristic.value();
        setRowToRotationMatrix(2, rot3);
    }

    // if base station's geometry is set then propagate new geometry
    for(uint8_t b = 0; b < baseStationsCount; b++) {
        if(!baseStations[b].isSet()) {
            return;
        }
    }
    stationCb(baseStations);
}
```

Ukázka kódu č. 18 Implementace metody *rotationWritten()* ve třídě *Bluetooth*.

6.1.5 Komponenta LedDriver

Tato třída ovládá RGB LED diodu bez blokování hlavního vlákna. Tento LED driver umožňuje zobrazit celkem 4 stavy, které mohou být libovolně nadefinovány, případně zhasnout všechny LED. Jednotlivé stavy jsou definovány jako pole struktury *LedTimings*, přičemž v této struktuře je definováno, kolik milisekund má

LED dioda svítit, kolik milisekund má být LED dioda zhaslá a GPIO pin dané LED diody. Definování jednotlivých stavů LED diod je uvedeno v ukázce kódu 19.

```
typedef struct {
    uint16_t turnOn;
    uint16_t turnOff;
    GPIO_PIN led;
} LedTimings;

const LedTimings failedTimings[] = {{500, 1000, PIN_RED_LED}};
const LedTimings trackingTimings[] = {{60, 940, PIN_BLUE_LED}};
const LedTimings trackingTimeoutTimings[] = {{80, 920, PIN_RED_LED}};
const LedTimings geometryNotSetTimings[] = {{60, 160, PIN_GREEN_LED},
             {160, 250, PIN_RED_LED}, {60, 1000, PIN_GREEN_LED}};
```

Ukázka kódu č. 19 Definování jednotlivých stavů LED diod.

Výše definované stavy LED diod jsou pro větší přehlednost uvedeny v následující tabulce 6.

Tabulka 6 Přehled definovaných stavů LED diod

Stav	Barva LED	Doba svícení [ms]	Doba zhasnutí [ms]
Failed	Červená	500	1000
Tracking	Modrá	60	940
Tracking timeout	Červená	80	920
Geometry not set*	Zelená	60	160
	Červená	160	250
	Zelená	60	1000

**Barvy se střídají postupně v pořadí od shora dolů*

Metoda *loop()* třídy *LedDriver* je volána v nekonečné smyčce v souboru main.cpp. V metodě *loop()* je nejprve podle aktuálního zobrazovaného stavu získán pointer na pole struktury *LedTimings* a délka tohoto pole. Dále je získána aktuální hodnota milisekund. Na začátku zobrazovaného cyklu je inicializována proměnná *startMillis* na hodnotu *currentMillis* a je nastavena hodnota proměnné *isLedTurnOn* na hodnotu *true*. Následuje rozhodování mezi aktuálním stavem LED diody. Pokud je LED dioda rozsvícená, je nastavena aktuálnímu GPIO pinu, ke kterému je

připojena LED dioda nízká logická úroveň, čímž dojde k rozsvícení dané LED diody. Následuje test, zda nesvítí LED dioda již příliš dlouhou dobu. Pokud ano, je LED dioda zhasnuta a je nastavena hodnota proměnné *startMillis* na aktuální hodnotu proměnné *currentMillis*. V případě, že má být LED dioda zhasnutá, je opět nejprve nastavena správná logická úroveň GPIO pinu, ke kterému je připojena aktuální LED dioda. Následuje opět porovnání délky zhasnutí LED, zda není zhaslá již příliš dlouho nad stanovený limit. Pokud ano, dojde k rozsvícení další LED diody v pořadí a je opět nastavena hodnota proměnné *startMillis* na hodnotu proměnné *currentMillis*. Implementaci celé metody *loop()* je možné vidět v ukázce kódu 20.

```
void LedDriver::loop(void) {
    const LedTimings *ledTiming;
    uint8_t totalTimings;
    switch(showingState) {
        case STATE_ALL_OFF: return;
        case STATE_FAILED:
            ledTiming = failedTimings;
            totalTimings = sizeof(failedTimings) / sizeof(*ledTiming);
            break;
        case STATE_TRACKING:
            ledTiming = trackingTimings;
            totalTimings = sizeof(trackingTimings) / sizeof(*ledTiming);
            break;
        case STATE_TRACKING_TIMEOUT:
            ledTiming = trackingTimeoutTimings;
            totalTimings = sizeof(trackingTimeoutTimings) / sizeof(*ledTiming);
            break;
        case STATE_GEOMETRY_NOT_SET:
            ledTiming = geometryNotSetTimings;
            totalTimings = sizeof(geometryNotSetTimings) / sizeof(*ledTiming);
            break;
        default: return;
    }
    uint32_t currentMillis = millis();
    if(!startMillis) { // the start of whole cycle
        startMillis = currentMillis;
        isLedTurnOn = true; }
    if(isLedTurnOn) { // LED is turned on
        digitalWrite(ledTiming[timingsIndex].led, LOW);
        if(currentMillis - startMillis > ledTiming[timingsIndex].turnOn) {
            isLedTurnOn = false; // turn off LED
            startMillis = currentMillis;
            digitalWrite(ledTiming[timingsIndex].led, HIGH);
        }
    } else { // LED is turned off
        digitalWrite(ledTiming[timingsIndex].led, HIGH);
        if(currentMillis - startMillis > ledTiming[timingsIndex].turnOff) {
            isLedTurnOn = true; // turn on LED
            startMillis = currentMillis;
            //end of cycle -> move index to next led timings
            timingsIndex = ++timingsIndex % totalTimings;
            digitalWrite(ledTiming[timingsIndex].led, LOW);
        }
    }
}
```

Ukázka kódu č. 20 Implementace metody *loop()* třídy *LedDriver*.

6.1.6 Komponenta TrackerGPIO

Jedná se o komponentu definující GPIO piny 3D trackeru a zároveň přetěžující tři funkce pro práci s GPIO piny z frameworku Arduino. Protože Arduinové desky běžně obsahují pouze 32 pinů na jednom portu, byl původní Arduino core pro nRF52840 napsán pouze pro GPIO piny na portu P0. 3D tracker však využívá i piny z portu P1, který není v Arduino core definován. Byly proto přetíženy původní Arduino funkce *pinMode()*, *digitalWrite()* a *digitalRead()*. Tyto funkce přijímají v argumentu pin jako strukturu *GPIO_PIN*, jenž obsahuje číslo portu a číslo pinu. Na začátku všech těchto třech funkcí je nejprve provedena kontrola, zda jsou validně nastaveny číslo portu i číslo pinu. Následně je vždy podle čísla portu zvolena počáteční adresa registru pro obsluhu GPIO pinů. Ukázky implementací těchto třech funkcí je možné vidět níže v ukázkách kódu 21, 22 a 23.

```
void digitalWrite(GPIO_PIN pin, uint32_t value) {
    if(!(pin.port == 0 || pin.port == 1) || pin.number > 31) {
        // nRF52840 has P0 or P1 with pin 0 - 31
        return;
    }

    switch (value) {
        case LOW:
            if(pin.port == 0) {
                NRF_GPIO->OUTCLR = (1UL << pin.number);
            } else {
                NRF_P1->OUTCLR = (1UL << pin.number);
            }
            return;

        default:
            if(pin.port == 0) {
                NRF_GPIO->OUTSET = (1UL << pin.number);
            } else {
                NRF_P1->OUTSET = (1UL << pin.number);
            }
            return;
    }
}
```

Ukázka kódu č. 21 Vlastní implementace funkce *digitalWrite()*.

```
void pinMode(GPIO_PIN pin, uint32_t mode) {
    if(!(pin.port == 0 || pin.port == 1) || pin.number > 31) {
        // nRF52840 has P0 or P1 with pin 0 - 31
        return;
    }
    switch(mode) {
        case INPUT:
            if(pin.port == 0) {
                NRF_GPIO->PIN_CNF[pin.number] =
       ((uint32_t)GPIO_PIN_CNF_DIR_Input << GPIO_PIN_CNF_DIR_Pos)
       | ((uint32_t)GPIO_PIN_CNF_INPUT_Connect    << GPIO_PIN_CNF_INPUT_Pos)
       | ((uint32_t)GPIO_PIN_CNF_PULL_Disabled    << GPIO_PIN_CNF_PULL_Pos)
       | ((uint32_t)GPIO_PIN_CNF_DRIVE_S0S1       << GPIO_PIN_CNF_DRIVE_Pos)
       | ((uint32_t)GPIO_PIN_CNF_SENSE_Disabled   << GPIO_PIN_CNF_SENSE_Pos);
            } else {
                NRF_P1->PIN_CNF[pin.number] =
       ((uint32_t)GPIO_PIN_CNF_DIR_Input        << GPIO_PIN_CNF_DIR_Pos)
       | ((uint32_t)GPIO_PIN_CNF_INPUT_Connect    << GPIO_PIN_CNF_INPUT_Pos)
       | ((uint32_t)GPIO_PIN_CNF_PULL_Disabled    << GPIO_PIN_CNF_PULL_Pos)
       | ((uint32_t)GPIO_PIN_CNF_DRIVE_S0S1       << GPIO_PIN_CNF_DRIVE_Pos)
       | ((uint32_t)GPIO_PIN_CNF_SENSE_Disabled   << GPIO_PIN_CNF_SENSE_Pos);
            }
            break;
        case INPUT_PULLUP:
            if(pin.port == 0) {...} else {...}
            break;
        case INPUT_PULLDOWN:
            if(pin.port == 0) {...} else {...}
            break;
        case OUTPUT:
            if(pin.port == 0) {...} else {...}
            break;
    }
}
```

Ukázka kódu č. 22 Vlastní implementace funkce *pinMode()*.

```
int digitalRead(GPIO_PIN pin) {
    if(!(pin.port == 0 || pin.port == 1) || pin.number > 31) {
        // nRF52840 has P0 or P1 with pin 0 - 31
        return 0;
    }

    if(pin.port == 0) {
        return ((NRF_GPIO->IN >> pin.number) & 1UL) ? HIGH : LOW;
    } else {
        return ((NRF_P1->IN >> pin.number) & 1UL) ? HIGH : LOW;
    }
}
```

Ukázka kódu č. 23 Vlastní implementace funkce *digitalRead()*.

6.2 Mobilní aplikace

Mobilní aplikace je cílená na chytré mobilní telefony s operačním systémem Android od verze 5.0, tj. API verze 21. Dále mobilní aplikace vyžaduje zařízení podporující OpenGL ES ve verzi 3.0, které je využíváno pro vykreslování trojrozměrné scény. Poslední požadavek kladený mobilní aplikací na zařízení je podpora Bluetooth Low Energy využívaného pro komunikaci s 3D trackerem. Aplikace obsahuje jednu hlavní aktivitu a je rozdělena do třech fragmentů. Jelikož je aplikace poměrně složitá, bude popsáno pouze několik zajímavých částí implementace.

6.2.1 Fragment Discovery

Discovery fragment obsahuje komponentu *RecyclerView*, skrze kterou jsou uživateli zobrazeny všechny objevené 3D trackery. Za skenování okolních 3D trackerů je zodpovědná třída *DevicesScanner*. Třída *DevicesScanner* dále zaobaluje komunikaci s 3D trackerem skrze BLE. Práce s Bluetooth je v Androidu poměrně komplikovaná, což vede k horší čitelnosti kódu a nepracuje se s ním příliš dobře. Třída *DevicesScanner* tedy slouží jako wrapper pro práci s BLE. Nabízí tak metody pro připojování, odpojování, skenování, zastavování skenování, odesílání geometrie 3D trackeru. Třída *DevicesScanner* dále využívá broadcastu pro odesílání BLE událostí všem komponentám aplikace. Ostatní komponenty aplikace si tak mohou zaregistrovat specifické zprávy o událostech, o kterých chtějí být informovány. *DeviceScanner* skenuje okolní bluetooth zařízení vždy maximálně 20 sekund.

Jelikož je veškerá práce s BLE asynchronní a na operace se musí čekat skrze callback metody, bylo pro komunikaci s 3D trackerem využito fronty, do které se umístí všechny operace čtení a zápis charakteristik. Po každé provedené operaci je tak vždy nutné vyčíst následující operaci nad charakteristikou z fronty a vykonat ji. Tato komunikace je nutná zejména pro čtení a nastavování geometrie 3D trackeru. Souřadnice z 3D trackeru z charakteristiky Coordinates jsou získávány pomocí mechanismu notify. Aby mohl 3D tracker odesílat data při každé změně do smartphonu, je nutné, aby se smartphone přihlásil k odběru notifikací charakteristiky Coordinates. Proto je nutné v Androidu povolit dané charakteristice notifikace skrze metodu *setCharacteristicNotification()* nad instancí *BluetoothGatt*. Dále je také nutné zapsat do deskriptoru s UUID „00002902-0000-1000-8000-00805f9b34fb“ dané charakteristiky hodnotu *0x01*,*0x00* označenou Androidem jako *ENABLE_NOTIFICATION_VALUE*. Metodu *subscribeNotifications()* sloužící pro přihlášení k odběru notifikací je možné vidět v ukázce kódu 24.

```
private void subscribeNotifications(
        BluetoothGattCharacteristic characteristic, BluetoothGatt gatt) {
    gatt.setCharacteristicNotification(characteristic, true);
    UUID uuid = UUID.fromString("00002902-0000-1000-8000-00805f9b34fb");
    BluetoothGattDescriptor descriptor =
                characteristic.getDescriptor(uuid);
    descriptor
        .setValue(BluetoothGattDescriptor.ENABLE_NOTIFICATION_VALUE);
    gatt.writeDescriptor(descriptor);
}
```

Ukázka kódu č. 24 Metoda *subscribeNotifications()* ze třídy *DevicesScanner*.

6.2.2 Fragment Geometry

Fragment Geometry obsahuje dvě komponenty *RecyclerView*. Tento fragment je zodpovědný za zobrazování obou geometrií uživateli, zobrazuje tedy uloženou geometrii a aktuální nastavenou geometrii prostřednictvím zmíněných *RecyclerView* komponent. Dále tento fragment umožňuje načítat geometrii z textového souboru a ukládat ji do interní paměti zařízení, odkud je pak tato geometrie vždy při startu aplikace načtena. Pro parsování textového souboru s geometrií, jenž je exportována skrze utilitu [19], využívá fragment Geometry implementaci *TextFileGeometryParser* rozhraní *GeometryParser*. Rozhraní *GeometryParser* obsahuje jednu metodu *parse()* přijímající v argumentu

InputStream a vracející list objektů *BaseStation*. Ve třídě *TextFileGeometryParser* je parsování prováděno po řádcích. Jakmile je parsování dokončeno, je takto získaná geometrie nastavena jako aktuální geometrie a je provedeno uložení této geometrie do interní paměti zařízení. K tomu je využíváno třídy *GeometryDAO*, skrze kterou je provedena serializace geometrie do formátu JSON. Tato třída je též používaná dále třídou *Geometry* pro deserializaci geometrie z formátu JSON z interní paměti zařízení. Třída *Geometry* obsahuje statickou proměnnou *stationGeometries*, ve které je uložený list objektů *BaseStation* představující aktuální geometrii. Deserializace geometrie do listu objektů *BaseStation* je prováděna vždy při startu aplikace skrze statickou metodu *init()* třídy *Geometry*. Třída *Geometry* též obsahuje metodu pro validaci geometrie.

6.2.3 Fragment Rendering

Tento fragment v sobě obsahuje jedinou *View* komponentu a tou je *SceneGLSurfaceView*. V této komponentě je při vytváření instance nejprve nastaven OpenGL ES kontext na verzi 3. Dále je vytvořena instance třídy *GLRenderer*, která je odpovědná za vykreslování trojrozměrné scény. Tato instance musí být nastavena skrze volání metody *setRenderer()* třídy *GLSurfaceView*. Dále je třída *SceneGLSurfaceView* zodpovědná za zachytávání touch událostí, které jsou potřeba pro ovládání kamery. Také je využíváno třídy *ScaleGestureDetector*, která detekuje standartní gesto pro změnu velikosti zoomu. Při vytváření instance třídy *ScaleGestureDetector* je třeba předat listener v parametru konstruktoru, pomocí kterého jsou získávány události o gestu zoomování. V listeneru je k dispozici metoda *onScale()*, přičemž v jejím parametru je předána instance třídy *ScaleGestureDetector*. Z tohoto detektoru lze získat koeficient pro zoom, který je poté předán rendereru.

GLRenderer vykresluje do scény osy souřadného systému, pokud jsou nastaveny geometrie Lighthouse stanic, tak jsou vykresleny obě Base Station jako kvádry o skutečných rozměrech a nakonec i všechny 3D trackery, které jsou připojeny a odesílají svou pozici. 3D trackery jsou vykresleny také jako kvádry o skutečných rozměrech. Dále je pro vykreslování využívána perspektivní projekční matice a kamera definovaná pomocí dvou úhlů azimut a zenit. Úhly azimut a zenit jsou získávány přepočtem ze souřadnic x, y z touch události. Výpočet azimutu a

zenitu se odvíjí od změny v souřadnicích *x*, *y* vypočtených jako *dx*, *dy*. Výsledná implementace listeneru touch události je uvedena v ukázce kódu 25.

```
@Override public boolean onTouchEvent(MotionEvent e) {
    scaleDetector.onTouchEvent(e);
    final int action = e.getActionMasked();
    switch (action) {
        case MotionEvent.ACTION_DOWN: {
            final int pointerIndex = e.getActionIndex();
            final float x = e.getX(pointerIndex);
            final float y = e.getY(pointerIndex);
            previousX = x; previousY = y;
            activePointerId = e.getPointerId(pointerIndex);
            break;
        }
        case MotionEvent.ACTION_MOVE: {
            final int pointerIndex = e.findPointerIndex(activePointerId);
            double newX = e.getX(pointerIndex);
            double newY = e.getY(pointerIndex);
            double dx = newX - previousX;
            double dy = newY - previousY;
            if(!scaleStarted)
                renderer.computeAngles(dx, dy);
            previousX = newX;
            previousY = newY;
            break;
        }
        case MotionEvent.ACTION_UP:
        case MotionEvent.ACTION_CANCEL: {
            activePointerId = INVALID_POINTER_ID;
            break;
        }
        case MotionEvent.ACTION_POINTER_UP: {
            final int pointerIndex = e.getActionIndex();
            final int pointerId = e.getPointerId(pointerIndex);
            if(pointerId == activePointerId) {
                final int newPointerIndex = pointerIndex == 0 ? 1 : 0;
                previousX = e.getX(newPointerIndex);
                previousY = e.getY(newPointerIndex);
                activePointerId = e.getPointerId(newPointerIndex);
            }
            break;
        }
    }
    return true;
}
```

Ukázka kódu č. 25 Implementace metody *onTouchEvent()* ze třídy *GLSurfaceView*.

Jelikož je potřeba otáčet s kamerou a zároveň měnit přiblížení kamery od počátku, je nutné správně implementovat listener touch událostí. Zachycená touch

událost musí být rovněž propagována *ScaleGestureDetectoru* zavoláním metody *onTouchEvent()* nad instancí detektoru. S kamerou je otáčeno v případě, že neprobíhá zoomování kamery přes *ScaleGestureDetector*. Dále je důležité správně inicializovat počáteční hodnoty proměnných *previousX* a *previousY*, protože uživatel používá pro manipulaci s kamerou více prstů. Využívá se tak pro inicializaci proměnných *previousX* a *previousY* vždy souřadnic pouze primárního prstu. Pokud by tomu tak nebylo, s každým dalším přiloženým prstem by kamera různě přeskakovala a byla by obtížně ovladatelná. Výsledné úhly azimut a zenit jsou vypočítány v metodě *computeAngles()* ve třídě *GLRenderer*. V metodě *computeAngles()* je nejprve vypočítán koeficient, kterým se násobí výsledné změny *dx*, *dy*. Výpočet koeficientu *multiplier* se odvíjí podle vzdálenosti kamery od počátku souřadného systému, kdy s větší vzdáleností je koeficient *multiplier* vyšší. Kód je možné vidět v ukázce kódu 26.

```
public void computeAngles(double dx, double dy) {
    double multiplier = (cam.getRadius() - MIN_CAMERA_ZOOM) *
            (MAX_ANGLE_MULTIPLIER - MIN_ANGLE_MULTIPLIER) /
            (MAX_CAMERA_ZOOM - MIN_CAMERA_ZOOM) + MIN_ANGLE_MULTIPLIER;

    dx *= multiplier;
    dy *= multiplier;

    double zenith = cam.getZenith() + Math.PI * (dy / height);
    double azimuth = cam.getAzimuth() + Math.PI * (dx / width);

    if(zenith > Math.PI / 2) zenith = Math.PI / 2;
    if(zenith < -Math.PI / 2) zenith = -Math.PI / 2;
    azimuth = azimuth % (2 * Math.PI);

    cam = cam.withAzimuth(azimuth).withZenith(zenith);
}
```

Ukázka kódu č. 26 Implementace metody *computeAngles()* ve třídě *GLRenderer*.

Kamera pozorovatele je z pohledu třetí osoby a je umístěna v počátku souřadného systému, tj. [0; 0; 0]. Kamera je otáčena kolem počátku ze vzdálenosti určené zoomem uživatele. Jak již bylo zmíněno výše, koeficient přiblížení je získáván ze třídy *ScaleGestureDetector*, přičemž výsledný rádius kamery je vypočten podle kódu uvedeného v ukázce 27.

```
public void zoom(float scale) {
    double scaleFactor = cam.getRadius() / scale;
    scaleFactor =
        Math.max(MIN_CAMERA_ZOOM, Math.min(scaleFactor, MAX_CAMERA_ZOOM));
    cam = cam.withRadius(scaleFactor);
}
```

Ukázka kódu č. 27 Metoda *zoom()* třídy *GLRenderer* vypočítávající přiblížení kamery.

Třída *GLRenderer* obsahuje implementaci broadcast recieveru skrze který jsou získávány informace o 3D trackerech a jejich nové souřadnice pozice. Pro vykreslování Base Station a 3D trackerů je využíváno grafických shaderů. Kvádry jak 3D trackerů, tak Base Station jsou definovány jako krychle o délce strany 1. Výsledné kvádry jsou získány vynásobením scale maticí. Kvádry Base Station jsou poté vynásobeny rotační maticí příslušné geometrie a nakonec posunuty na pozici dané geometrie. Kvádry 3D trackerů jsou vždy pouze posunuty z počátku na aktuální souřadnice daného 3D trackeru v prostoru. Při implementaci shaderových programů bylo vypočteno i osvětlení pomocí Blinn-Phongova osvětlovacího modelu. Výsledné kódy vertex shaderu a fragment shaderu pro krychli jsou uvedeny v ukázkách kódu 28 a 29.

```
#version 300 es

in vec3 inPosition;
in vec3 inNormal;

uniform mat4 matM;
uniform mat4 matV;
uniform mat4 matP;
uniform vec4 lightPosition;

out vec3 normal;
out vec3 lightDir;
out vec3 viewDir;
out float lightDistance;

void main() {
   vec4 pos = matM * vec4(inPosition, 1.0);
   vec4 posMV = matV * pos;
   gl_Position = matP * posMV;
   normal = inverse(transpose(mat3(matV * matM))) * inNormal;
   normal = normalize(normal);
   lightDir = (lightPosition - pos).xyz;
   lightDistance = length(lightDir);
   viewDir = -posMV.xyz;
}
```

Ukázka kódu č. 28 Vertex shader program pro vykreslování krychle.

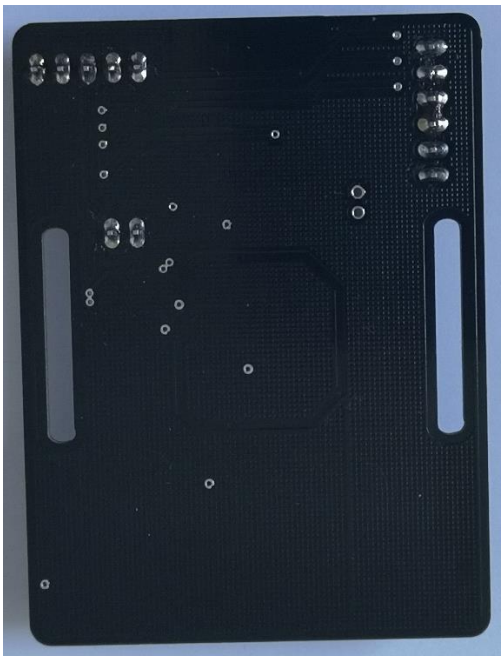
```
#version 300 es

precision highp float;

in vec3 normal;
in vec3 lightDir;
in vec3 viewDir;
in float lightDistance;

uniform vec4 color;

out vec4 outColor;

vec4 calculateLight() {
    vec4 baseColor = vec4(color);
    vec4 ambient = vec4(0.2, 0.2, 0.2, 1.0);
    vec4 specular = vec4(0.5, 0.5, 0.5, 1.0);

    vec3 ld = normalize(lightDir);

    float NDotL = max(0.0, dot(ld, normal));
    vec3 halfVector = normalize(lightDir + viewDir);
    float NDotH = max(0.0, dot(halfVector, normal));

    vec4 totalAmbient = baseColor * ambient;
    vec4 totalDiffuse = NDotL * baseColor;
    vec4 totalSpecular = specular * pow(NDotH, 12.0);

    return totalAmbient + totalDiffuse + totalSpecular;
}

void main() {
    outColor = calculateLight();
}
```

Ukázka kódu č. 29 Fragment shader program pro vykreslování krychle.

7 Výsledky a testování

Vlastní 3D tracker kompatibilní s HTC Vive se podařilo úspěšně vytvořit. Byly zhotoveny celkem dva 3D trackery, přičemž kromě Bluetooth modulů je k dispozici materiál na výrobu dalších třech trackerů. Vytvořený 3D tracker je schopný v reálném čase trackovat vlastní pozici. 3D tracker je uzpůsoben jako wearable zařízení, má tak tedy kompaktní rozměry a deska obsahuje dva otvory pro uchycení pásku. Díky tomu je tedy možné využít pásek pro umístění 3D trackeru na ruku. Rozměry celého 3D trackeru činí pouhých 6,3x4,7 cm. Zhotovený 3D tracker je možné vidět na obrázku 20 a 21.

Obrázek 20 Vrchní strana hotového 3D trackeru.



Obrázek 21 Spodní strana hotového 3D trackeru.

Napájet 3D tracker je možné dvěma způsoby. První možností je napájení pomocí mikro USB konektoru. Druhým způsobem napájení 3D trackeru je možnost napájení z jedné 3 V lithiové baterie CR2032. Při napájení z baterie byla otestována celková doba provozu 3D trackeru. Celková doba provozu, kdy tracker nepřetržitě odesílal data o své poloze se pohybovala něco přes 3 hodiny.

Dále byla otestována funkčnost automatického přepínání mezi napájením z mikro USB konektoru a 3 V baterií. Toto automatické přepínání funguje naprosto

bez problémů, přepínání je velmi rychlé a nedochází ani k restartování celého zařízení.

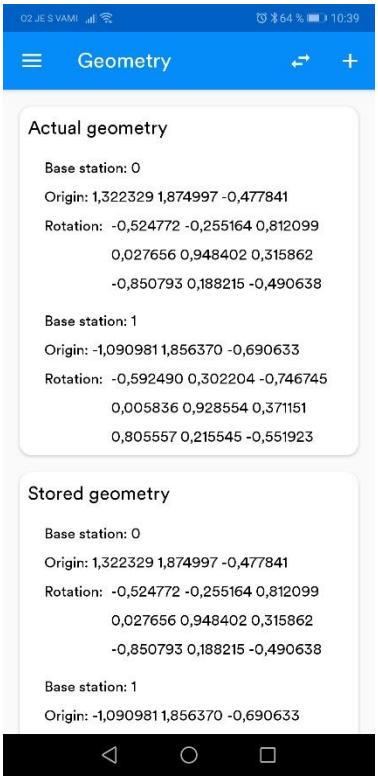
Schopnost trackovat pozici 3D trackerem je možné s přesností na jeden milimetr. Jedná se tak tedy o vysoce přesné trackování pozice. Jediným úskalím je využití pouze jedné fotodiody, kdy může lehce dojít k jejímu zastínění a 3D tracker poté není na okamžik schopen trackovat současnou pozici. Nicméně vzhledem k tomu, že byla snaha o vytvoření co nejlevnějšího a nejmenšího zařízení, se s tímto nedostatkem předem počítalo.

Dále byla vytvořena mobilní aplikace pro operační systém Android od verze 5.0. Aplikace disponuje českou a anglickou lokalizací. Tato aplikace je určena jak pro celkové ovládaní, tak i pro samotné uživatele, kterým umožní sledovat pozici 3D trackeru. Aplikace byla proto rozdělena do dvou uživatelských režimů, kdy se jedná o uživatelský a administrátorský režim. Administrátorský režim je navíc obohacen o práci s geometrií, která je nedílnou součástí potřebnou pro trackování pozice 3D trackeru. Získanou geometrii pomocí utility [19] je možné z textového souboru uložit do interní paměti zařízení a využívat ji tak pro nastavování 3D trackerům. Administrátorský režim je možné aktivovat kliknutím 10krát za sebou na ikonu aplikace v dialogu O aplikaci.

Jedním z hlavních úkolů aplikace je pak vykreslování pozice 3D trackerů v prostoru. Aplikace tak vykresluje obě Base Station podle jejich aktuální pozice a natočení v prostoru, dále všechny 3D trackery, ke kterým je smartphone aktuálně připojen a počátek souřadného systému, ve kterém jsou umístěny osy. Celá scéna je vykreslována podle skutečného měřítka, kdy délka osy činí 1 metr.

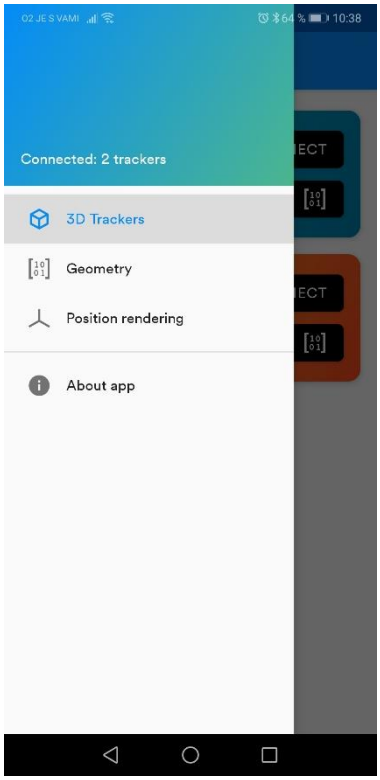
Veškeré screenshoty obrazovek z aplikace jsou provedeny v administrátorském režimu aplikace. Snímky z výsledné aplikace je možné vidět na obrázcích 22 – 27.

Obrázek 22 Fragment 3D trackery, ve kterém jsou spravovány veškeré trackery.

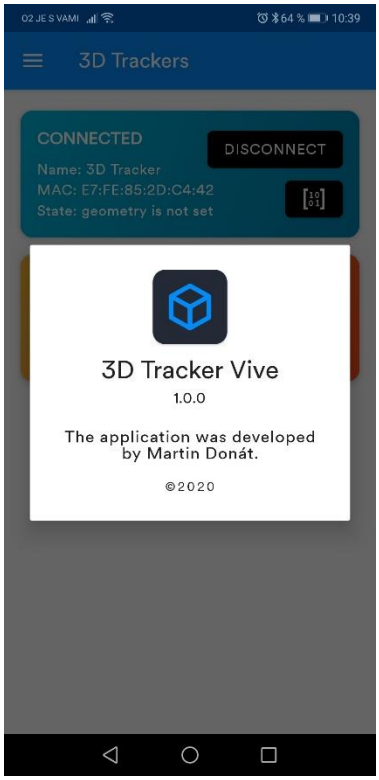


Obrázek 23 Fragment Geometrie, ve kterém je možné vidět obě geometrie.

Na obrázku 22 je možné vidět výsledný fragment 3D trackery. V tomto fragmentu je provedeno skenování okolních 3D trackerů, možnost připojení, odpojení a posílání geometrie. Na obrázku 23 je fragment Geometrie, který zobrazuje obě geometrie, tedy aktuální a uloženou geometrii. Skrze tento fragment je možné importovat geometrii z textového souboru a případně nastavit uloženou geometrii jako aktuální.

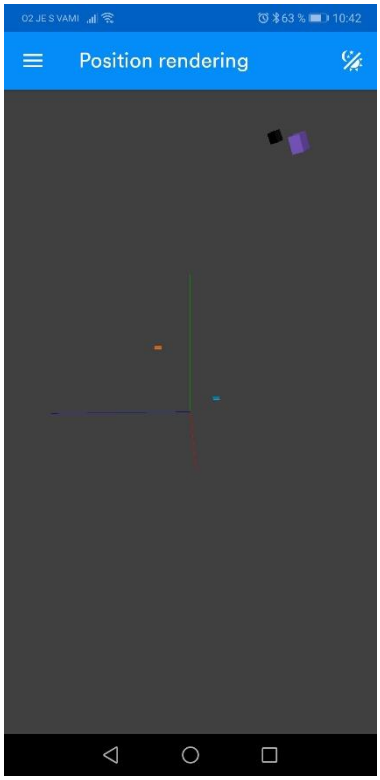


Obrázek 24 Hlavní menu aplikace.

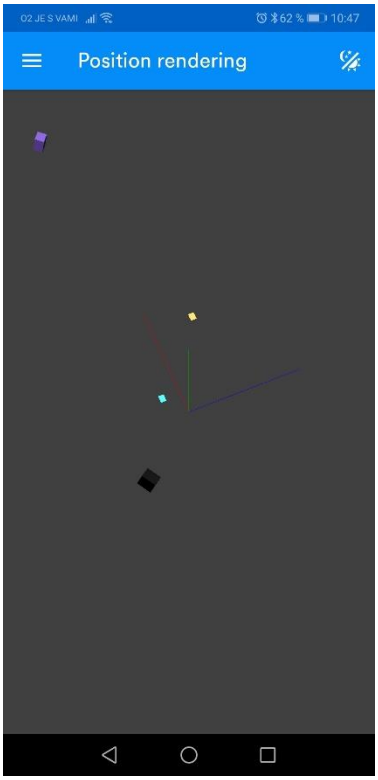


Obrázek 25 Dialog O aplikaci.

Menu celé aplikace je možné vidět na obrázku 24. V menu je zobrazeno například také kolik je aktuálně připojených trackerů. Na obrázku 25 je vidět dialog O aplikaci, skrze který je možné aplikaci přepnout do administrátorského režimu.



Obrázek 26 Příklad fragmentu Vykreslování pozice s natočením kamery 1.



Obrázek 27 Příklad fragmentu Vykreslování pozice s natočením kamery 2.

Na obrázcích 26 a 27 je možné vidět fragment Vykreslování pozice. Takto se pozadí nachází ve tmavém režimu, ikonka v aplikační liště však také umožňuje přepnout pozadí do světlého režimu. Je možné vidět dva 3D trackery – modrý a oranžový a dvě Base Station, kdy fialová představuje master a černá představuje slave.

8 Závěr

Cílem diplomové práce bylo navrhnout a implementovat 3D tracker kompatibilní s HTC Vive. Cíl práce se podařilo úspěšně splnit, 3D tracker byl přes veškeré překážky úspěšně zhotoven. Všechny zdrojové kódy, včetně souborů návrhu hardwaru jsou volně dostupné ve dvou GitHub repozitářích [21, 22].

3D tracker je schopný trackovat vlastní pozici v reálném čase s přesností na jeden milimetr. Co se týče napájení, nabízí se dvě možnosti, a to buď využití mikro USB konektoru, nebo 3 V lithiové baterie CR2032 se kterou je 3D tracker schopen pracovat přibližně 3 hodiny. Vzhledem k tomu, že byl od počátku 3D tracker navrhován jako wearable zařízení, je možné tracker nosit na ruce pomocí pásku jako hodinky.

Jako součást práce byla naprogramována také mobilní aplikace pro operační systém Android umožňující celkové ovládání 3D trackerů. Aplikace dále umožňuje sledování pozice 3D trackeru v prostoru. Mobilní aplikace je rozdělena do dvou režimů, běžného uživatelského a administrátorského, kdy v administrátorském režimu je umožněna i práce s geometrií, která je pro trackování naprosto nezbytná. Hlavním úkolem aplikace je pak vykreslování 3D trackerů v prostoru, kdy aplikace vykresluje všechny připojené 3D trackery, obě Base Station a počátek souřadného systému, ve kterém jsou umístěny osy.

9 Seznam použité literatury

[1] Our History. *Bluetooth Technology Website* [online]. [vid. 2020-01-27]. Dostupné z: https://www.bluetooth.com/about-us/our-history/

[2] Join the SIG. *Bluetooth Technology Website* [online]. [vid. 2020-01-27]. Dostupné z: https://www.bluetooth.com/develop-with-bluetooth/join/

[3] DONÁT, Martin. *Vývoj periferie s využitím Bluetooth Low Energy SoC* [online]. B.m., 2018 [vid. 2020-01-27]. Bakalářská práce. Univerzita Hradec Králové, Fakulta informatiky a managementu. Dostupné z: https://theses.cz/id/690508/?isshlret=Martin%3BDon%C3%A1t%3B;zpet=%2Fvyhledavani%2F%3Fsearch%3Dmartin%20don%C3%A1t%26start%3D1

[4] Radio Versions. *Bluetooth Technology Website* [online]. [vid. 2020-01-27]. Dostupné z: https://www.bluetooth.com/learn-about-bluetooth/bluetooth-technology/radio-versions/

[5] WHEATLEY, Richard. *A view on Bluetooth Low Energy stack roles | NXP Community* [online]. 8. březen 2018 [vid. 2020-01-30]. Dostupné z: https://community.nxp.com/thread/332319

[6] nRF5 SDK v16.0.0. *Nordic Semiconductor Infocenter* [online]. 3. leden 2020 [vid. 2020-01-27]. Dostupné z: https://infocenter.nordicsemi.com/topic/struct_sdk/struct/sdk_nrf5_latest.html

[7] SoftDevices. *Nordic Semiconductor Infocenter* [online]. 24. leden 2020 [vid. 2020-01-27]. Dostupné z: https://infocenter.nordicsemi.com/topic/struct_nrf52/struct/nrf52_softdevices.html

[8] *nRF52840 - Nordic Semiconductor* [online]. [vid. 2020-01-28]. Dostupné z: https://www.nordicsemi.com/Products/Low-power-short-range-wireless/nRF52840

[9] All you need to know about SteamVR Tracking 2.0... will it be the foundation of Vive 2? *The Ghost Howls* [online]. 7. červen 2017 [vid. 2020-01-29]. Dostupné z: https://skarredghost.com/2017/06/07/need-know-steamvr-tracking-2-0-will-foundation-vive-2/

[10] LANG, Ben. Latest HTC Vives Are Shipping with Tweaked Base Stations, Redesigned Packaging. *Road to VR* [online]. 13. duben 2017 [vid. 2020-04-15]. Dostupné z: https://www.roadtovr.com/latest-vive-shipping-with-tweaked-base-stations-redesigned-packaging/

[11] NAIROL. *Light Emissions* [online]. C. 2020 [vid. 2020-04-16]. Dostupné z: https://github.com/nairol/LighthouseRedox/blob/master/docs/Light%20Emissions.md

[12] NAIROL. *Base Station Info Block* [online]. C. 2020 [vid. 2020-04-16]. Dostupné z: https://github.com/nairol/LighthouseRedox/blob/master/docs/Base%20Station.md#base-station-info-block

[13] TRENDEL, Simon. *A framework for motion analysis of anthropomimetic robots* [online]. B.m., nedatováno. Diplomová práce. Technische Universität München, Department of Informatics. Dostupné z: https://cdn.hackaday.io/files/19570837282880/thesis2.pdf

[14] *VIVE™ | VIVE Virtual Reality System* [online]. 2020 2011 [vid. 2020-01-30]. Dostupné z: https://www.vive.com/us/product/vive-virtual-reality-system/

[15] BASILEUS-PHOENIX. *Updated Smartphone Render by basileus-phoenix on DeviantArt* [online]. 3. březen 2015 [vid. 2020-01-30]. Dostupné z: https://www.deviantart.com/basileus-phoenix/art/Updated-Smartphone-Render-517746664

[16] HTC Vive Teardown. *iFixit* [online]. 26. duben 2016 [vid. 2020-04-16]. Dostupné z: https://www.ifixit.com/Teardown/HTC+Vive+Teardown/62213

[17] SHTUCHKIN, Alexander. *Position calculation in detail · ashtuchkin/vive-diy-position-sensor Wiki* [online]. 28. listopad 2016 [vid. 2020-04-16]. Dostupné z: https://github.com/ashtuchkin/vive-diy-position-sensor/wiki/Position-calculation-in-detail

[18] SUNDAY, Dan. *Distance between Lines, Segments and their CPA (2D & 3D)* [online]. [vid. 2020-04-16]. Dostupné z: http://geomalgorithms.com/a07-_distance.html#Distance-between-Lines

[19] SHTUCHKIN, Alexander. *ashtuchkin/vive-diy-position-sensor-geometry-getter* [online]. C++. 2019 [vid. 2020-04-17]. Dostupné z: https://github.com/ashtuchkin/vive-diy-position-sensor-geometry-getter

[20] TS4231. *Triad Semiconductor* [online]. [vid. 2020-04-11]. Dostupné z: https://www.triadsemi.com/product/ts4231/

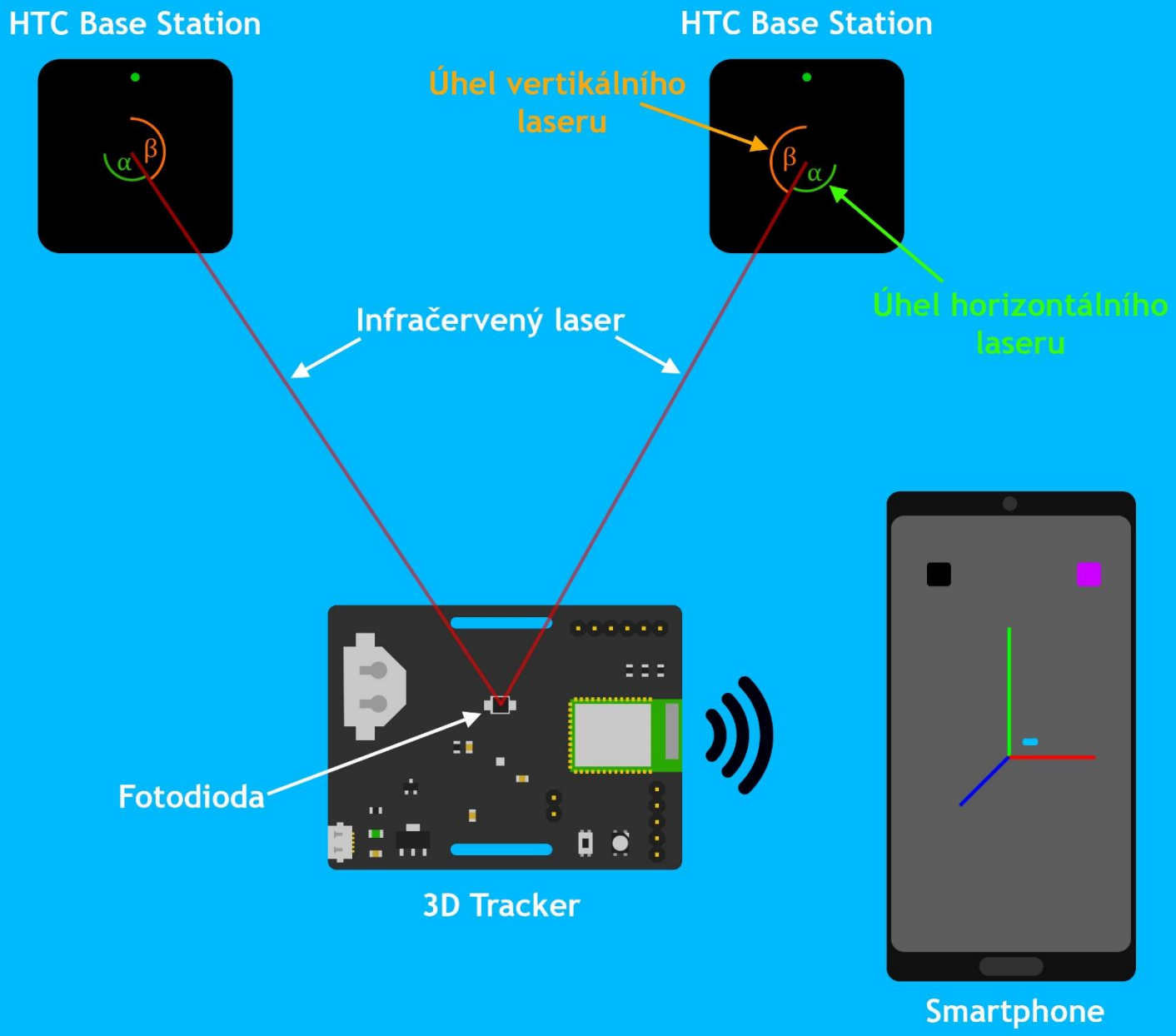
[21] DONÁT, Martin. *kacer/3D-Tracker-Vive* [online]. C++. 2020 [vid. 2020-04-21]. Dostupné z: https://github.com/kacer/3D-Tracker-Vive

[22] DONÁT, Martin. *kacer/3D-Tracker-Vive-Android* [online]. Java. 2020 [vid. 2020-04-21]. Dostupné z: https://github.com/kacer/3D-Tracker-Vive-Android

10 Přílohy

1) Ve složce přílohy se nachází:
    a. Složka poster obsahuje vyexportovaný poster ve formátu JPEG a originální soubor PSD pro editaci.
    b. Složka 3D tracker obsahuje Eagle soubory, Gerber soubory a zdrojové kódy firmwaru.
    c. Ve složce mobile app se nachází APK soubor s aplikací a zdrojové kódy mobilní aplikace.
2) Vědecko-popularizační poster znázorňující principy celého řešení 3D trackeru.

3D Tracker kompatibilní s HTC Vive



HTC Vive

Jedná se o set určený pro **virtuální realitu** neboli VR. V tomto setu se nachází headset (náhlavní souprava), ovladače pro obě ruce a dvě zařízení Base Station. HTC Vive využívá pro trackování pozice technologii **SteamVR tracking** vyvinutou společností Valve. Tato technologie je unikátní oproti technologiím, které používají ostatní VR sety. Ostatní technologie využívají pro trackování pozice kamer, zatímco SteamVR využívá **infračervených paprsků**. Díky tomu je technologie SteamVR daleko přesnější a zároveň méně náročná na výpočetní výkon.

3D tracker

Tohle malé vytvořené zařízení dokáže **trackovat** (sledovat) svoji pozici v prostoru s využitím HTC Vive Base Station a tu poté skrze **Bluetooth** odeslat do smartphonu. Aby však smartphone mohl přijímat souřadnice 3D trackeru, musí se k danému 3D trackeru připojit. Jakmile je Bluetooth spojení navázáno, **mobilní aplikace vykreslí** aktuální pozici 3D trackeru v trojrozměrné scéně.

Princip funkce

Trackování pozice 3D trackeru v prostoru je prováděno vůči relativní pozici obou **Base Station**. Base Station jsou takové majáky, které postupně **vrhají laserové paprsky** do místnosti. Vrhání paprsků je prováděno v cyklech, kdy v každém cyklu je vrhán pouze jeden paprsek jednou stanicí. Vrhá se nejprve ve směru horizontálním, potom vertikálním, přičemž se Base Station **střídají**. Vždy před vrháním laserového paprsku obě Base Station prosvítí celou místnost **resetovacím** infračerveným pulzem. 3D tracker je tak schopný přepočítat dobu mezi resetovacím infračerveným světlem a zásahem laseru na **úhel**.

Poté co jsou známy všechny **čtyři úhly**, je možné vypočítat vektory směřující od obou Base Station k **fotodiodě** 3D trackeru. Pokud jsou známy pozice obou Base Station a jejich rotace v prostoru, je možné vypočítat **průsečík** těchto dvou vektorů. Jakmile je vypočítán průsečík, jsou známy **souřadnice 3D trackeru** v prostoru. A je to!

UNIVERZITA HRADEC KRÁLOVÉ
Fakulta informatiky a managementu
Akademický rok: 2018/2019

Studijní program: Aplikovaná informatika
Forma studia: Prezenční
Obor/kombinace: Aplikovaná informatika (ai2-p)

Podklad pro zadání DIPLOMOVÉ práce studenta

Jméno a příjmení:	**Bc. Martin Donát**
Osobní číslo:	**I1800106**
Adresa:	Týnišťská 1197, Třebechovice pod Orebem, 50346 Třebechovice pod Orebem, Česká republika
Téma práce:	Vývoj 3D trackeru kompatibiního s HTC Vive
Téma práce anglicky:	Development of 3D tracker compatible with HTC Vive
Vedoucí práce:	Ing. Pavel Kříž, Ph.D. Katedra informatiky a kvantitativních metod

Zásady pro vypracování:

Cíl: S použitím vhodného Bluetooth Low Energy chipu navrhnout a implementovat prostorový tracker kompatibiní s HTC Vive. Součástí bude poster prezentující principy a celé řešení, který bude vhodný např. pro Den otevřených dveří, Noc vědců a jiné vědecko-popularizační akce.
Osnova:

1. Úvod
2. Cíl
3. Technologie Bluetooth Low Energy (BLE) na platformě chipů Nordic Semiconductor
4. 3D tracking pomocí zařízení HTC Vive
5. Analýza a návrh řešení vlastního trackeru
6. Implementace
7. Výsledky a testování
8. Závěr

Seznam doporučené literatury:

1. https://www.nordicsemi.com/Products/Low-power-short-range-wireless/nRF52840
2. https://www.instructables.com/id/Sub-10-DIY-Vive-Tracker/
3. https://github.com/ashtuchkin/vive-diy-position-sensor
4. https://hackaday.com/2018/09/04/this-is-your-solution-for-open-source-motion-tracking/ https://hackaday.io/project/19570-htc-vive-lighthouse-custom-tracking
5. https://github.com/solinvictus21/Zippies/tree/master/LighthouseCircui

Podpis studenta: Datum:

Podpis vedoucího práce: Datum: