VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV POČÍTAČOVÉ GRAFIKY A MULTIMÉDIÍ
DEPARTMENT OF COMPUTER GRAPHICS AND MULTIMEDIA

INTERIÉROVÁ LOKALIZACE VE 3D

3D INDOOR LOCALIZATION

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE — **Bc. JIŘÍ ZVĚŘINA**
AUTHOR

VEDOUCÍ PRÁCE — **Ing. VLADIMÍR VESELÝ, Ph.D.**
SUPERVISOR

BRNO 2018

Abstrakt

V této práci se zabývám návrhem a tvorbou 3D vizualizace interiérové lokalizace s využitím systému od společnosti Sewio Networks s.r.o. Vizualizace je zprostředkována skrze webové prostředí nad technologií WebGL. K usnadnění práce je využita knihovna Three.js. Pro lepší rozšířitelnost a budoucí vývoj aplikace rozebírám způsoby propojení vizualizace pomocí Three.js s architekturou Model-View-Controller. Dále se zabývám využitím senzorů k pořízení dodatečných dat a vylepšení lokalizace. Konkrétně se jedná o senzory k měření tlaku s jejichž pomocí jsem schopen odhadnout vertikální pozici lokalizovaných zařízení a senzory inerciální, které poskytují informace o natočení sledovaných zařízení.

Abstract

This thesis is about designing and creating 3D visualisation of interior localization using Sewio system. Visualisation is presented as web application with WebGL technology. WebGL deployment is simplified with Three.js library. For future scalability of the application, I deal with integration of Three.js into Model-View-Controller architecture. The last part of the thesis is about additional sensors and their use to make the localization more precise. There is a barometric senzor making vertical localization possible and inertial sensors useful for determining rotation of localized devices.

Klíčová slova

Sewio, WebGL, JavaScript, Three.js, MVC, senzory, web

Keywords

Sewio, WebGL, JavaScript, Three.js, MVC, sensors, web

Citace

ZVĚŘINA, Jiří. *Interiérová lokalizace ve 3D.* Brno, 2018. Diplomová práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Vladimír Veselý, Ph.D.

Interiérová lokalizace ve 3D

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením Ing. Vladimíra Veselého, Ph.D. Potřebné informace mi poskytli Ing. Lubomír Mráz a Ing. Tomáš Kočan. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

.....
Jiří Zvěřina
23. května 2018

Poděkování

Tímto děkuji Ing. Vladimíru Veselému, Ph.D., že se ujal vedení mé práce a dohlížel na její průběh. Dále děkuji Ing. Lubomíru Mrázovi za poskytnutí příležitosti pracovat na této práci pro společnost Sewio Networks s.r.o. a konzultace ohledně toho, kterým směrem se má tato práce ubírat. Děkuji Ing. Tomáši Kočanovi za rychlé zasvěcení do problematiky a za veškeré technické rady, které mi poskytl. Děkuji Michaele Tučkové za gramatickou revizi mé práce. Nakonec také děkuji Jakubu Duchoňovi, MSc., za komentáře, kterými odlehčoval náročné dny strávené u této práce.

Jako dodatek k poděkování přikládám recept na Gbelskou fazolovou polévku mé babičky. Fazoli propláchneme a namočíme do vody alespoň na 6 hodin. Po odmočení ji dáme vařit společně se zeleninou (mrkev, petržel, cibule) a přidáme bujón (pokud nechceme používat bujón, použijeme vývar). Jakmile fazole změkne, přidáváme do polévky nakrájenou pálivou klobásu (např. čabajku) a utřený česnek, čímž polévce dodáme základ chuti. Připravíme si na oleji cibulovou zásmažku. Zásmažku dochutíme sladkou i pálivou paprikou, majoránkou, a pokud jsme nepoužili příliš čerstvého česneku, můžeme přidat také sušený. Zásmažkou polévku zahustíme. Po přidání zásmažky polévku dochutíme a chvíli povaříme. Polévka se podává s domácími nudlemi, které připravíme z vajíčka, vody, soli, hrubé a hladké mouky (ve stejném poměru). Nudle připravujeme zvlášť. K polévce je vhodné nudle nakrájet na širší kusy.

Obsah

Kapitola 1

Úvod

Velkým trendem poslední doby je sbírání a analýza dat. Tímto se také zabývá společnost Sewio Network s.r.o, dále jen Sewio. Konkrétně se jedná o lokalizační data v interiérech. Získaná data se dají interpretovat různým způsobem. Společnost Sewio doposud zobrazovala data pouze ve dvourozměrném prostoru, čímž omezovala jejich potenciál. Tím se také dostávám k této práci, jejímž cílem je získané informace zobrazovat ve třech rozměrech prostřednictvím webového prohlížeče bez nutnosti instalace jakéhokoliv dalšího softwaru. Dále se budu věnovat možnostem vylepšení těchto dat za pomoci přídavných senzorů (barometr, magnetometr, gyroskop).

V kapitole 2 se budu zaobírat technologií lokalizačního systému. Popíšu již zmíněné senzory, jakým způsobem mohou vylepšit lokalizaci, jaké informace mohou poskytnout a jak tyto informace využít. Mimo jiné také rozeberu webové technologie, které byly v práci použity.

Návrh aplikace jako takové bude popsán v kapitole 3. Budu se zaobírat tím, jak je aplikace koncipována z pohledu návrhových vzorů a zaměřím se na jejich propojení s použitými webovými technologiemi. Mimo jiné popíšu způsob komunikace aplikace s technologií společnosti Sewio a návrh přenášených dat.

Kapitola 4 bude shrnovat, postup implementace 3D vizualizační aplikace a řešení problémů s ní souvisejících. Také je v kapitole zaznamenáno jak funguje serverová část systému společnosti Sewio, kvůli následující implementaci výpočtu vertikální lokalizace s pomocí barometrických senzorů, jehož velká část se odehrává právě v části serveru. Nakonec popíši právě výpočet výšky zařízení a to, jak byly v systému data synchronizovány, aby výpočet byl vůbec možný.

Testování je středem zájmu kapitoly 5, ve které představuji především naměřené hodnoty a vyvozuji závěry o tom, jak je řešení použitelné. Testování je rozděleno na část, kde se zabývám funkčností řešení a na část, kde se zabývám právě použitelností řešení.

V poslední řadě vše shrnuji v kapitole 6. V této kapitole je popsáno čeho bylo v práci dosaženo, jaký přínos tato práce přináší a také se zaobírám možnostmi navázání na tuto práci v budoucnosti pro dosažení lepších výsledků a vytvoření nových poznatků.

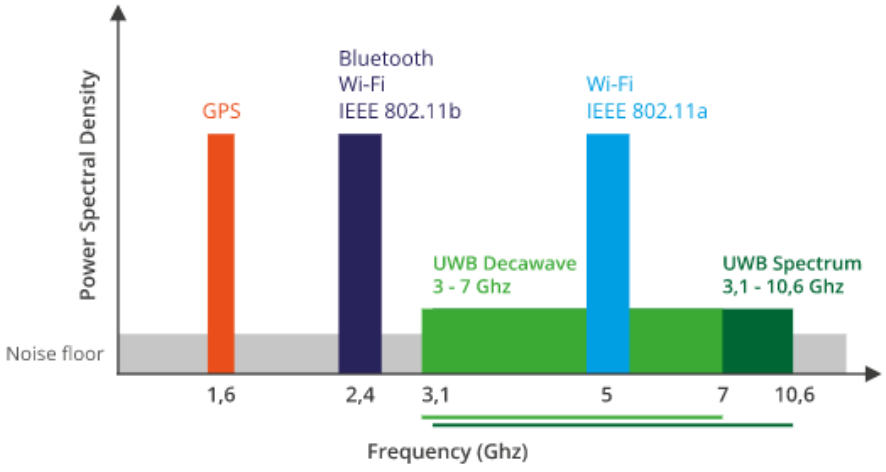
Kapitola 2

Lokalizace ve 3D

Pro nastínění situace v této kapitole popíši základ technologie Sewio a technologie, na kterých je založena. Popíši senzory, s jejichž pomocí budu zísaná data vylepšovat, a také popíšu webové technologie využité pro 3D vizualizaci a přenos dat.

2.1 Sewio

Technologie společnosti Sewio je založena na možnostech přenášení dat pomocí takzvaného UWB, neboli Ultra Wide Band. Jedná se o bezdrátovou technologii využívající široké frekvenční spektrum. Porovnání UWB s ostatními technologiemi ilustruje obrázek 2.1. Její vlastnosti a také nízká energetická náročnost ji činí vhodnou pro aplikaci lokalizace, jak v laboratorních podmínkách, tak v reálných industriálních prostorách [9]. Lokalizaci umožňuje kombinace dvou typů senzorů.



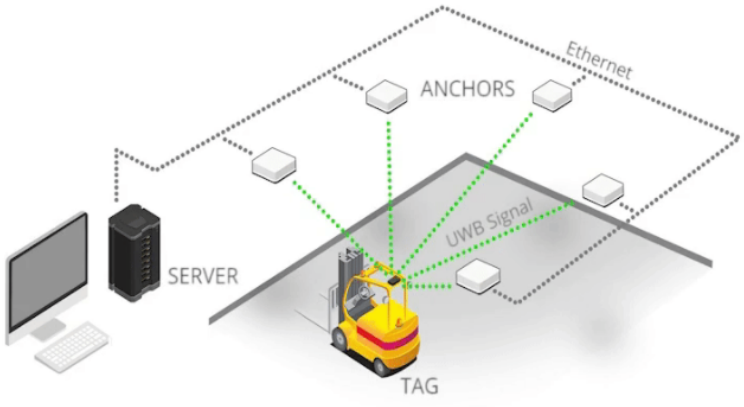
Obrázek 2.1: Porovnání bezdrátových technologií. Zdroj: `https://rtlsuwb.com/uwb`

Jedním typem je zařízení označované jako **tag**. Toto zařízení v pravidelných intervalech (v základním nastavení) vysílá identifikační informaci. Dále se o tomto zařízení zmíním v sekci Senzory.

Druhým typem zařízení je **anchor**, neboli **kotva**. Těchto kotev je nutné mít umístěno několik ideálné v místech, kde jednotlivé kotvy vidí na sebe (z důvodu jejich vzájemné

synchronizace). Tato zařízení přijímají signál z tagů a pomocí TDOA (Time Difference of Arrival) jsou na serveru počítány přibližné polohy míst, ze kterých byly signály odeslány [5]. Tato technika bude v práci dále rozvedena, protože je klíčová pro optimalizace lokalizace s využitím barometrů.

Data mezi kotvami a serverem jsou přenášena standardně pomocí drátové technologie Ethernet (IEEE 802.3), popřípadě bezdrátově pomocí Wi-Fi (IEEE 802.11). Bezdrátový přenos zde přináší několik komplikací v rámci zmíněné synchronizace kotev. Příklad rozložení technologie Sewio ilustruje obrázek 2.2.

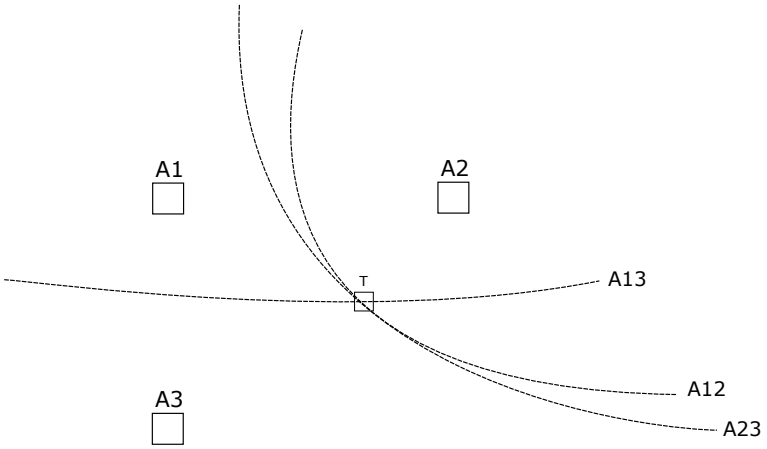


Obrázek 2.2: Základní rozložení pro lokalizaci technologií Sewio. Zdroj: `https://www.sewio.net/rtls-tdoa/`

2.2 TDOA

Technologii TDOA, neboli Time Difference of Arrival popíši na reálném případě využití technologií Sewio. Pro představu rozložení využiji opět obrázek 2.2. Tag (na obrázku připěvněn na střeše vysokozdvižného vozíku) vysílá signál s nějakým identifikačním číslem (zelené čáry). Signál je zachycen na kotvách, u kterých je přesně definována jejich pozice v rámci systému. Čas přijetí signálu s tímto identifikačním číslem je na kotvách zapsán a odeslán na server. Logicky tedy kotvy, které jsou k tagu blíže, signál zachytí dříve. Čas bude mít pro tyto kotvy menší hodnotu než u kotev, které jsou od tagu dále. Pro zjednodušení budu předpokládat, že signál přichází pouze jednou a není nikde odražen. S těmito problémy je nutné se v reálné situaci vypořádat. Také je potřeba podotknout, že pro možnost výpočtu multilaterace je třeba mít kotvy alespoň tři.

Všechny možné polohy bodu, vypočtené dle naměřených časů na každých dvou kotvách, vytvářejí hyperboloidy. Jelikož systém nepočítá se třemi rozměry, jsou tyto výpočty promítány do 2D prostoru; jedná se tedy o hyperboly, jak ilustruje obrázek 2.3. Pokud všechny hyperboly necháme protnout, získáme výsledný bod (v reálném případě spíše množinu bodů), který odpovídá reálné pozici, ze které byl signál odeslán [5]. Vylepšení projekce s využitím výšky naměřené z barometrů, může poskytovat lepší výsledky lokalizace.



Obrázek 2.3: Ukázka přibliženého výpočtu multilaterace.

2.3 Senzory

K získání přesnějších, popřípadě nových informací o sledovaných objektech, jsou k tagu přidány speciální senzory. Většina těchto senzorů má využití ve 3D lokalizaci, z toho důvodu právě senzory v této sekci popíšu.

2.3.1 Pohybové senzory

Mezi pohybové senzory řadím akcelerometry, které slouží výhradně k určení, jestli je tag ve stacionární poloze, či nikoliv. Akcelerometry jsou dobrým prostředkem pro šetření baterie tagu a snížení vytíženosti komunikačního pásma. Mimo jiné mohou akcelerometry sloužit také ke zlepšení přesnosti určování polohy, pokud jsou využity v kombinaci s ostatními senzory.

2.3.2 Rotační senzory

Jednou z informací, které mohou být uživateli poskytovány skrze 3D vizualizaci, je natočení tagu v prostoru. Tato informace může být využita například pro určení toho, kterým směrem se tag bude dále pohybovat, čímž může být dále zpřesňována výsledná poloha tagu.

Informaci o natočení tagu poskytuje dvojice senzorů. Jedná se o gyroskop, který měří úhlové zrychlení okolo tří ortogonálních os tagu. Informace z tohoto senzoru je však pouze relativní. Ke správnému natočení tagu je potřeba ještě magnetometru, který měří sílu magnetického pole opět ve třech ortogonálních osách. Kombinací těchto senzorů jsme schopni určit přesné natočení tagu podle vertikální osy vůči magnetickému severu Země. Natočení podle horizontálních os je v tomto případě relativní vzhledem ke startovací poloze tagu (poloha ve které je provedena kalibrace). Obrázek 2.4 popisuje natočení tagu podle ortogonálních os.

Popis rotace tagů může mít několik podob. Nejzákladnější reprezentací rotace jsou Eulerovy úhly. Pokročilejší a také v mnoha ohledech lepší reprezentací jsou quaterniony.

Obrázek 2.4: Ortogonální osy pro určení rotace tagu. Zdroj: `https://www.sewio.net/product/li-ion-tag-imu-tdoa/`

Eulerovy úhly

Rotace popsána pomocí Eulerových úhlů je reprezentována třemi úhly. Tyto úhly budu označovat jako α, β a γ. Pokud původní osy prostoru označíme jako x, y a z a výsledné osy jako X, Y a Z. Eulerovy úhly mají následující parametry:

- α - úhel mezi osou x a vektorovým součinem $z \times Z$;
- β - úhel mezi osou z a výslednou osou Z;
- γ - úhel mezi vektorovým součinem $z \times Z$ a osou x;

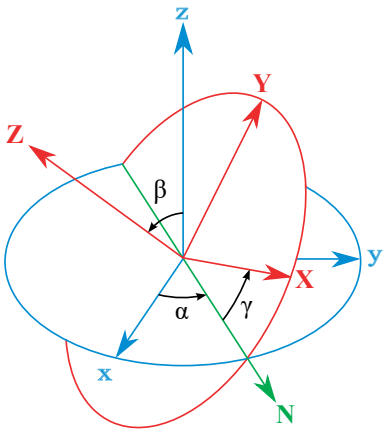
Pomocí těchto tří úhlů můžeme jednoduše definovat finální natočení objektu resp. tagu tak, jak to demonstruje obrázek 2.5. Tato reprezentace bohužel přináší několik omezení. Prvním omezením je takzvaný Gimbal lock. Jedná se o kombinaci Eulerových úhlů ve které není možné otáčet výsledné těleso všemi směry. Ke Gimbal locku dochází, pokud se jeden úhel dostane do kritických hodnot a ostatní dva úhly v tu chvíli poskytují stejnou informaci natočení [3]. Těchto kombinací je několik a reprezentace pomocí Eulerových úhlů se kvůli nim stává pro mé použití nevhodná. Druhým omezením je nepřesná interpolace mezi dvěma natočeními definovanými pomocí Eulerových úhlů. Interpolovaný úhel je možné počítat lineárně pomocí následující rovnice:

$$\delta = \alpha_0 + step * (\alpha_1 - \alpha_0) \tag{2.1}$$

,kde δ je výsledný úhel a *step* je krok interpolace v rozmezí $\langle 0, 1 \rangle$.

Aplikace Eulerových úhlů je standardně ve 3D grafice prováděna násobením matic reprezentujících pozici vertexů ve scéně s maticemi reprezentující jejich finální natočení [3]. Matice reprezentující natočení mají tvar:

$$\begin{aligned} M_z(\delta) &= \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\delta) & -\sin(\delta) \\ 0 & \sin(\delta) & \cos(\delta) \end{bmatrix}, \\ M_y(\delta) &= \begin{bmatrix} \cos(\delta) & 0 & -\sin(\delta) \\ 0 & 1 & 0 \\ \sin(\delta) & 0 & \cos(\delta) \end{bmatrix}, \\ M_x(\delta) &= \begin{bmatrix} \cos(\delta) & -\sin(\delta) & 0 \\ \sin(\delta) & \cos(\delta) & 0 \\ 0 & 0 & 1 \end{bmatrix} \end{aligned} \tag{2.2}$$



Obrázek 2.5: Demonstrace využití Eulerových úhlů. Zdroj: `https://en.wikipedia.org/wiki/Euler_angles`

Výsledná rotační rovnice je vytvořena násobením těchto matic následovně:

$$M = M_z(\alpha)M_y(\beta)M_x(\gamma) \tag{2.3}$$

Quaterniony

Quaterniony jsou prvky z oboru hyperkomplexních čísel popsané Williamem Rowanem Hamiltonem. Quaterniony jsou vhodné pro definici rotací ve 3D prostoru, jelikož oproti Eulerovým úhlům u nich nedochází k již popsaném Gimbal locku [7]. Další výhodou popisu rotace tělesa s využitím quaternionů je redukce chyby naskládané při násobení čísel s desetinnou čárkou, kde při využití popisu rotační maticí dochází také k parazitnímu zvětšení měřítka objektů a zkosení. Rotace s využitím quaternionů mohou být reprezentovány v následující podobě:

$$a + bi + cj + dk \tag{2.4}$$

,kde a, b, c a d jsou reálná čísla a i, j, k jsou quaternionové jednotky. Pro použití ve 3D grafice se zavádí notace:

$$q = (q_0, q_1, q_2, q_3) \tag{2.5}$$

Quaternion je zde tedy prvek z $\mathbb{R}^4$ [3].

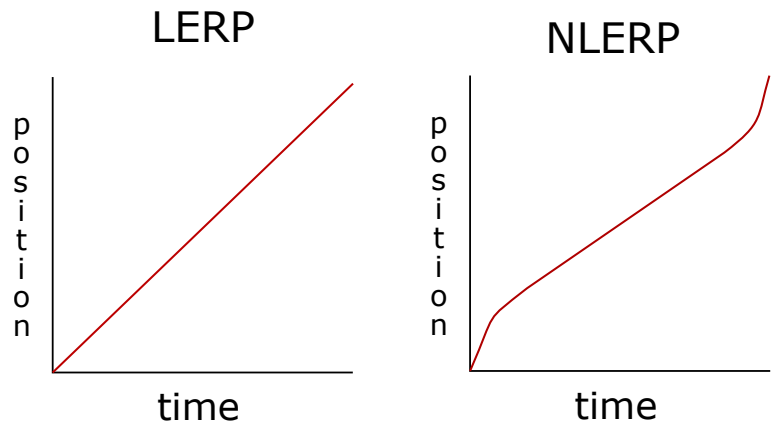
Rotační matice s využitím quaternionů má tvar:

$$M(q) = \begin{bmatrix} q_0^2 + q_1^2 - q_2^2 - q_3^2 & 2q_1q_2 + 2q_0q_3 & 2q_1q_3 - 2q_0q_2 \\ 2q_1q_2 - 2q_0q_3 & q_0^2 - q_1^2 + q_2^2 - q_3^2 & 2q_2q_3 + 2q_0q_1 \\ 2q_1q_3 + 2q_0q_2 & 2q_2q_3 - 2q_0q_1 & q_0^2 - q_1^2 - q_2^2 + q_3^2 \end{bmatrix} \tag{2.6}$$

Pro plynulý přechod mezi dvěma quaterniony není standardní lineární interpolace dostačující. Je potřeba využít takzvané normalizované sférické lineární interpolace, neboli NLERPu. NLERP dvou quaternionů lze vypočítat rovnicí:

$$NLERP(|q_1|, |q_2|, step) = |q_1|(|q_1|^{-1}|q_2|)^{step} \tag{2.7}$$

Nevýhodou je, že objekt interpolovaný pomocí NLERPu se pohybuje rychleji na začátcích a koncích interpolace, což demonstruje obrázek 2.6 [6].



Obrázek 2.6: Porovnání křivek pohybu při LERPu (vlevo) a NLERPu (vpravo).

2.3.3 Tlakové senzory

Barometry umístěné na tagu i kotvách mohou sloužit k určení výškových rozdílů mezi těmito zařízeními. Jelikož jsou polohy kotev předem definovány v systému, lze pomocí barometru určit přesnou pozici podél osy Z. Barometrů lze také využít k určování patra u vícepodlažních budov, kde jsou systémy umístěny přímo nad sebou. Bez barometrů je toto určování složitý problém, jelikož signál z tagů může putovat i mezi více podlažími.

Výpočet rozdílu výšky mezi referenčním zařízením (u kterého výšku známe) a zařízením u kterého výšku hledáme se skládá z následujících kroků:

1. Měření tlaku P_R na referenčním zařízení.
2. Měření tlaku P_X a teploty T (termometr je také součástí senzoru) na zařízení u něhož není známá výška.
3. Výpočet rozdílu výšky ΔH, pomocí hypsometrické rovnice: $\Delta H = \frac{RT}{g} \cdot ln(\frac{P_X}{P_R})$ (R je molární plynová konstanta a g je lokální gravitační zrychlení).

Z rozdílu výšky můžeme určit finální hodnotu výšky zařízení jednoduchým rozdílem se známou výškou referenčního zařízení [11]. Nevýhodou používání barometru k určení výšky zařízení je vysoká náchylnost na náhlé změny tlaku v místnosti, které mohou být způsobeny například otevřením okna, spuštěním klimatizace, popřípadě fouknutím na některé ze zařízení. Takovéto změny tlaku mohou mít za následek chybu ve výpočtu polohy i o několik metrů. Těmto situacím se dá předcházet vhodným filtrováním dat.

Vzorec pro výpočet tlaku je možné dále zjednodušit. Existuje varianta, která se jmenuje Babinetův vzorec, která poskytuje méně přesné výsledky pro rozdíly výšky přesahující 1000 metrů, což se ale v našem případě nemůže stát, proto je tento vzorec vhodný pro použití v našich podmínkách. Rovnice 2.8 popisuje výpočet rozdílu výšek [1].

$$\Delta z = 16000(1 + 0.004t_m)\frac{p_0 - p_1}{p_0 + p_1} \tag{2.8}$$

Filtrování tlakových dat

Situace, kdy dojde k náhlé změně tlaku v místnosti jsou krátké a rychle dochází k vyrovnání naměřeného tlaku mezi zařízeními. V ostatních případech by vypočítané hodnoty

měly správně kopírovat realitu. Jelikož jsou tyto skoky krátké, mohou být řešením těchto problémů vyhlazovací filtry. Ve většině těchto případů bude pravděpodobně stačit exponenciální klouzavý průměr. Výhodou je, že k dispozici nejsou data pouze z jednoho referenčního zařízení, ale hned z několika. Pokud se vychýlí hodnoty pouze na jednom z nich, můžeme předpokládat, že došlo k chybě a měření tohoto zařízení zanedbat.

Exponenciální klouzavý průměr je typ klouzavého průměru u nějž jsou starší hodnoty ohodnoceny exponenciálně nižší váhou. Můžeme ho počítat rekurzivně pomocí:

$$S_t = \begin{cases} Y_1, & t = 1 \\ \alpha Y_t + (1-\alpha) S_{t-1}, & t > 1 \end{cases} \tag{2.9}$$

, kde α je vyhlazující faktor mezi 0 a 1. Y_t je hodnota v čase t. S_t je hodnota klouzavého průměru v čase t [8].

2.4 Webové technologie

Jedním z trendů poslední doby je migrace standardních desktopových aplikací do webového prostředí, což umožňuje uživatelům používat aplikace, bez toho aniž by si je museli předem instalovat, popřípadě aniž by potřebovali jakékoliv další předinstalované závislosti vyjma webového prohlížeče, který je součástí všech počítačů.

Tohoto trendu se drží také aplikace zobrazující informace ve třech rozměrech. S příchodem API WebGL[1] je možné na webových prohlížečích zobrazovat 3D grafiku akcelerovaně. WebGL je grafické API založené na OpenGL ES[2], ze kterého přebírá mnoho funkcí. Současný stav vestavěných systémů doposud neumožňuje vyrovnat se zobrazovacími schopnostmi a výkonem klasickému OpenGL[3], u kterého je v posledních verzích možné využívat například technik globální iluminace [21]. Pomalu se ale propast mezi těmito dvěma technologiemi zmenšuje.

2.4.1 Three.js

Knihovna Three.js[4] zaobaluje funkce WebGL a vytváří uživatelsky přívětivé možnosti práce s 3D grafikou. Jelikož je knihovna navržena spíše pro tvorbu menších aplikací a ukázek, je nutné řešit, jakým způsobem knihovnu využít pro tvorbu mohutnější aplikace. Řešením těchto potíží se zabývám v kapitole 3.

Protože je knihovna grafickým základem aplikace popíši některé teoretické principy toho jak tato knihovna funguje, respektive jak využít její části k dosažení nejlepších výsledků po výkonnostní stránce.

Geometrie

Pokud chceme vizualizovat jakýkoliv objekt pomocí grafické karty, musíme nejprve definovat tvar takovéhoto objektu. Všechny objekty jsou popsány trojicemi bodů v prostoru neboli vertexy. Každá trojice vytváří polygon, což je jedna ploška výsledného objektu [19]. Pokud chceme plochu definovat čtyřmi body, musíme vytvořit polygony dva. Knihovna Three.js

[1] `https://www.khronos.org/webgl/`
[2] `https://www.khronos.org/opengles/`
[3] `https://www.khronos.org/opengl/`
[4] `https://threejs.org/`

nabízí k využití dva typy geometrií. Původním typem je *Geometry*, který je více uživatelsky přívětivý, ale nenabízí stejný výkon jako modernější *BufferGeometry*.

S využitím *BufferGeometry* je snížen počet přenosů mezi grafickou kartou a procesorem, jelikož tato struktura odpovídá tomu, jak jsou data uložena na grafické kartě. Vertexy a jejich indexy jsou zde tedy uloženy jako atributy [16]. Tento přístup sice komplikuje následnou práci s geometrií, ale pro scény s velkým počtem objektů s danou geometrií rapidně zvyšuje výkon aplikace.

Materiály

Každý objekt má kromě své geometrie také svůj materiál. Materiál definuje to jak bude objekt ve scéně renderován. Svým způsobem materiál odpovídá tomu, jaký shader bude použit pro vykreslení výsledného objektu. Nejzákladnější rozdíl mezi jednotlivými shadery je v tom, jaký osvětlovací model je využit k výpočtu výsledné barvy konkrétního pixelu. Volba materiálu zde však nemá za následek pouhou změnu výpočtu, ale je jí ovlivněna celá pipeline grafické karty, tedy i to jak je nahlíženo na jednotlivé vertexy uvnitř vertex shaderu, popřípadě to jestli jsou na vertexy aplikovány transformace uvnitř geometry shaderu, pro výsledný efekt daného materiálu [2].

2.4.2 Ostatní knihovny

Jelikož je výsledkem práce webová aplikaci, není nutné kreslit všechny části grafického uživatelského rozhraní přímo do WebGL kontextu. HTML samo o sobě nabízí široké možnosti pro tvorbu a stylování tlačítek a dalších prvků rozhraní. Knihovny, které jsou vhodné pro tvorbu takovéhoto rozhraní jsou jQuery[5] a dat.gui[6]. Knihovna jQuery slouží k lepší manupulaci s HTML Document Object Model. Knihovna dat.gui umožňuje vytvářet jednoduché ovládací prvky pro aplikaci.

[5] `https://jquery.com/`

[6] `http://workshop.chromeexperiments.com/examples/gui/1--Basic-Usage`

Kapitola 3

Návrh aplikace

Aplikace je navržena jako součást webového balíčku RTLS Studio[1] společnosti Sewio. Jedná se o 3D webovou aplikaci nad technologií WebGL pro akcelerované zobrazování 3D grafiky v internetovém prohlížeči. Aplikace by měla kopírovat vizuální styl, kterým se ubírají poslední aplikace RTLS Studia.

3.1 Architektura aplikace

Jádro aplikace je postavené na architektuře Model View Controller, zkráceně MVC. Základ vizuální stránky tvoří HTML jádro s CSS soubory, definujícími podstatné části vzhledu kontrolních panelů webové aplikace. Hlavní část aplikace je rozdělena do JavaScriptových souborů rozdělených podle tříd a funkce v architektuře MVC. Kód v JavaScriptu využívá ECMAScript 6, který přidává ke standardnímu JavaScriptu několik moderních funkcí, jako jsou například nové definice tříd a jejich konstruktorů, popřípadě nové typy proměnných *let* (dočasná proměnná) a *const* (konstantní proměnná) [17]. ECMAScript 6 je z větší části podporován všemi moderními webovými prohlížeči.

3.1.1 MVC

Existuje mnoho architektur pro návrh aplikace interagujících s uživatelem skrze uživatelské rozhraní. Jedním z nich je MVC. Hlavní myšlenkou této architektury je oddělit data od samotného GUI aplikace. Architektura rozděluje, jak již z názvu vyplývá, kód aplikace na tři části. Každá z těchto částí má svoji úlohu v systému [10]. Výhodou takovéhoto přístupu je dobrá rozšířitelnost a znovupoužitelnost mnohých částí aplikace.

View

View se skládá čistě z komponent grafického uživatelského rozhraní, jako jsou například tlačítka, nebo textová pole. Uživatelovy interakce jsou z View předávány Controlleru, kde jsou zvoleny odpovídající metody, které mají být provedeny.

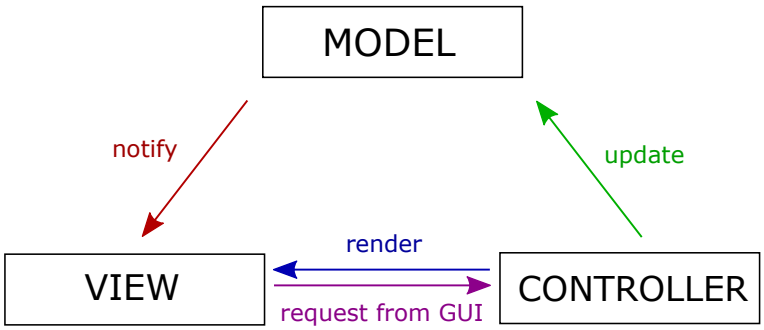
Model

Model obsahuje definici objektů a dat, které tyto objekty obsahují. Měl by zaobalovat všechny informace, které známe. Data může upravovat pouze Controller.

[1] `https://www.sewio.net/rtls-studio-sw/`

Controller

Controller řídí aplikaci a spojuje Model a View. Pro každou akci uživatele zachycenou na View je v Controlleru zvolena metoda, která upravuje data v Modelu, čímž se aktualizuje View. Architekturu je možné popsat obrázkem 3.1.



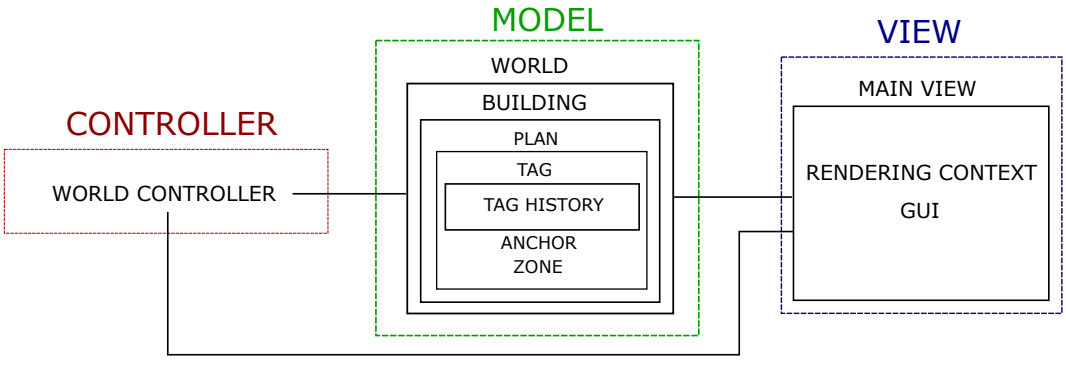
Obrázek 3.1: Popis architektury MVC.

Integrace architektury MVC

Veškeré části aplikace, které mají co dočinění s grafickým uživatelským rozhraním, nebo knihovnou Three.js, budou zahrnuty v části View. Jedná se o:

- renderování scény;
- zachycení uživatelského vstupu a převedení na funkce obsažené v Controlleru;
- udržení synchronizace zobrazovaných objektů s jejich modely;

Všechny části systému nesoucí informaci, jako jsou tagy a kotvy, popřípadě plány budov, budou v části Model. V Controlleru budeme reagovat na vstupy a budeme aktualizovat Model [12]. Controller je nahraditelná část aplikace. Pokud budeme chtít změnit funkcionalitu některého z tlačítek uživatelského rozhraní namapovaného na konkrétní funkci Controlleru, stačí nahradit tuto funkci v Controlleru. Obrázek 3.2 demonstruje konkrétní návrh tříd aplikace v rámci MVC architektury.



Obrázek 3.2: Rozdělení tříd podle architektury MVC.

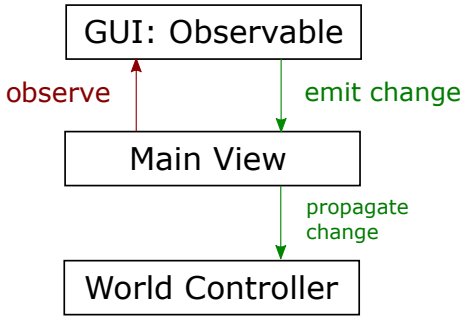
3.1.2 Návrhové vzory

Žádná škálovatelná aplikace se neobejde bez návrhových vzorů. Velmi užitečným návrhovým vzorem využitelným v jakékoliv aplikaci nad architekturou MVC je Observer. Pro potřeby tvorby objektů s využitím knihovny Three.js je výhodné použít návrhový vzor Mediator.

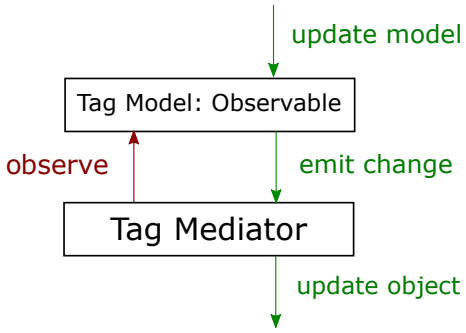
Observer

Tento návrhový vzor slouží k upozornění na změnu hlídané hodnoty objektu. Výhodou je, že hlídaný objekt si nemusí uchovávat informaci o tom, které objekty změnu hlídají.

V mé aplikaci má observer dvě funkce. První funkcí je předávání informací mezi View a Controllerem. Druhou předávání informací mezi Modelem a jeho vizualizovanou částí. Funkce ilustrují obrázky 3.3 a 3.4.



Obrázek 3.3: Komunikace View-Controller s využitím observeru.

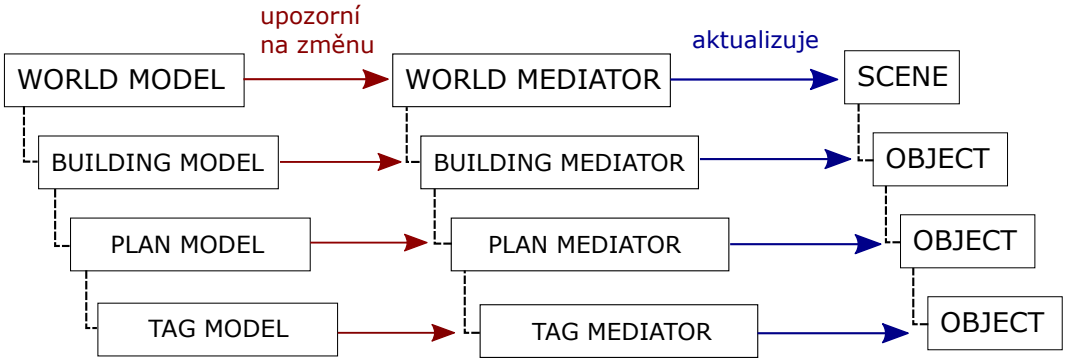


Obrázek 3.4: Komunikace Model-View s využitím observeru.

Mediator

Mediator plní funkci jakéhosi prostředníka mezi objekty. Objekt komunikuje s mediatorem (v mém případě notifikací observeru), který za něj rozhodne co se má stát.

Konkrétně v aplikaci mediátoři slouží jako prostředníci mezi modelem objektů (tag, kotva) a jejich vizualizovanou formou v Three.js. Jak toto propojení funguje, ilustruje obrázek 3.5. Volbu vhodného mediátoru určuje podle názvu třídy modelu tovární metoda.



Obrázek 3.5: Ukázka využití mediatoru v aplikaci.

3.2 Komunikace

Aby aplikace mohla správně vizualizovat reálnou situaci, musí z nějakého vnějšího zdroje čerpat data. K získávání dat jsou v aplikaci využity dvě metody.

První metodou je získávání statických dat pomocí REST API společnosti Sewio. Data jsou rozdělena hierarchicky podle toho, co v reálu zaobalují. Jelikož může být lokalizace realizována v rámci většího množství budov, budova je největší možný celek. Z části MVC vyplývá, že jsem vytvořil ještě větší datový model svět, pod který spadají všechny budovy. Toto rozdělení je vhodné pro mapování na strukturu scény knihovny Three.js, kde jsou objekty také takto hierarchicky seřazeny. Pro každou budovu je nutné načíst všechny plány, které budova obsahuje. Jedná se o lokalizované místnosti, popřípadě celá patra budovy. Nyní už je možné načítat tagy, kotvy a zóny, které jsou roztříděny podle toho, na kterém plánu se nacházejí. Dotazy na REST API se z Javascriptového prostředí provádějí skrze XHR. Této metody je možné také využít pro odesílání 3D modelů načtených aplikací na server.

Druhou metodou je získávání dynamických dat obsahujících současný stav prvků načtených pomocí předchozí metody. K této komunikaci slouží technologie WebSocketů. K získávání těchto dat je třeba se nejprve přihlásit. API společnosti Sewio je navrženo tak, aby bylo možné přihlásit se k odběru prvků pomocí ID, které bylo předem získáno z RESTu. Druhou metodu můžeme uložit do služby, která bude komunikovat s modely dynamických prvků.

3.2.1 Ukládání modelů na server

Aplikace vizualizující lokalizaci pomocí 3D grafiky by měla umožňovat načítat vlastní modely, lépe reprezentující objekty, popřípadě osoby, které jsou lokalizovány. Je důležité předem navrhnout, kam se takovéto modely na serveru budou ukládat a jak k nim bude moci aplikace opětovně přistupovat.

Nejprve je potřeba model správně uložit. Existuje mnoho formátů pro ukládání 3D modelů s různými vlastnostmi. Pro potřeby Three.js je načtený model interpretován jako geometrie skládající se z mnoha vertexů. Tyto vertexy můžou být buďto seřazeny tak, aby tvořily polygony samy o sobě, nebo je tato informace uvedena v poli indexů (tato varianta je paměťově méně náročná). Jelikož je formát geometrie pro potřeby aplikace univerzální a dá se převádět na řetězec zvolil jsme ho jako vhodnou reprezentaci 3D modelu.

Z hlediska uložení popsané reprezentace 3D modelu se nabízí dvě možnosti. První je uložit soubor s modelem na disk serveru a do databáze zanést informaci o tom, kde se

tento soubor nachází a jaké má vlastnosti. Nevýhodou tohoto přístupu je nutnost přidávat všechny tyto soubory k databázi, pokud chceme přenést server na jiný počítač. Tato operace se u serverů společnosti Sewio provádí poměrně často, zvolil jsem proto druhou variantu. Touto variantou je uložení kompletního modelu jako řetězce přímo do databáze. Existuje zde několik omezení na velikost přenášených souborů. Tyto omezení se nicméně dají upravit v konfiguračních souborech, což pro většinu modelů zobrazovaných aplikací není ani nutné.

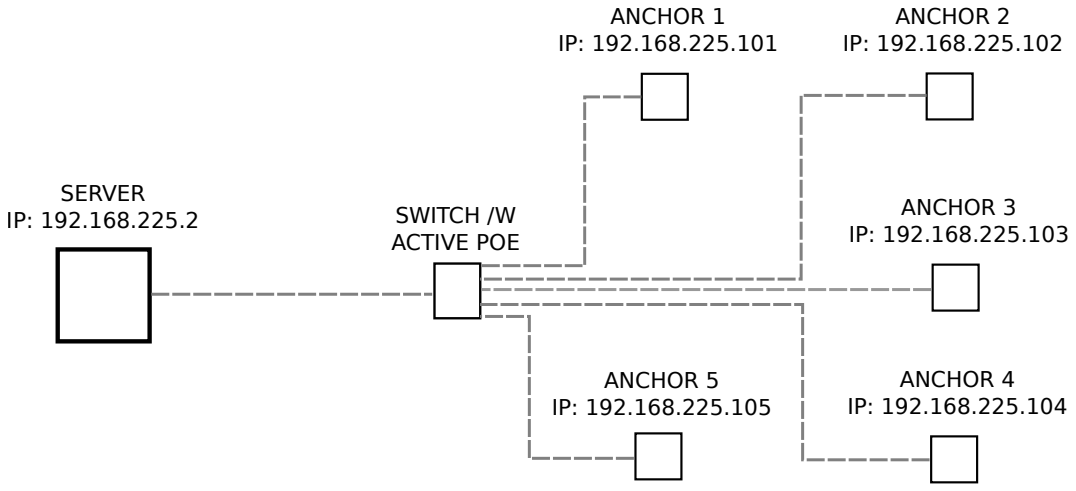
Kapitola 4

Implementace

Kapitola implementace je rozdělena do tří oddělených částí. První část je o rozložení systému společnosti Sewio tak, aby bylo možné získávat potřebná data k lokalizaci. Druhá část se týká programování 3D vizualizace. Ve třetí části se zabývám úpravou serveru, aby byl schopen počítat s barometrickými daty poskytnutými lokalizačními zařízeními.

4.1 Rozložení systému

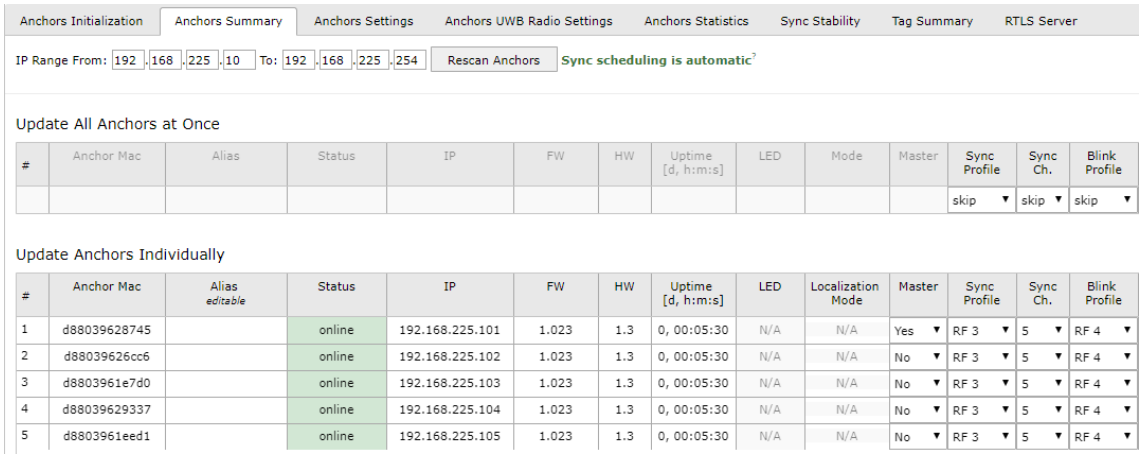
Jak již bylo nastíněno v předešlých kapitolách, existují dva typy zařízení používané pro lokalizaci. Každé z těchto zařízení má několik variant. K tomu abychom mohli pracovat s inerciálními a barometrickými senzory musíme využít tagy typu IMU s verzí firmwaru 3.118 a vyšší. Dále je k lokalizaci možné použít kotvy ve verzi 3.002, nastavené na bezdrátový, nebo standardní přenos. Pokud zvolíme přenos bezdrátový, musíme na kotvách nastavit přístupový bod, ke kterému máme připojený server. Mimo jiné, pokud nepoužíváme technologii DHCP a kotvy jsou k serveru připojeny například skrze switch, je nutné každé kotvě nastavit unikátní IP adresu. Rozložení kotev, které jsem zvolil pro účely tvorby aplikace a následnému testování, demonstruje obrázek 4.1.



Obrázek 4.1: Rozložení kotev pro účely testování.

Jestliže máme systém připraven po hardwarové stránce, je nutné mít na serveru nainstalován softwarový balíček RTLS Studio. Pokud je vše nastaveno správně, program RTLS

Manager, běžící na serveru jako služba, je schopen kotvy a tagy identifikovat. Na obrázku 4.2 je viditelná identifikace jednotlivých zařízení RTLS Managerem.



#	Anchor Mac	Alias	Status	IP	FW	HW	Uptime [d, h:m:s]	LED	Localization Mode	Master	Sync Profile	Sync Ch.	Blink Profile
1	d88039628745		online	192.168.225.101	1.023	1.3	0, 00:05:30	N/A	N/A	Yes	RF 3	5	RF 4
2	d88039626cc6		online	192.168.225.102	1.023	1.3	0, 00:05:30	N/A	N/A	No	RF 3	5	RF 4
3	d8803961e7d0		online	192.168.225.103	1.023	1.3	0, 00:05:30	N/A	N/A	No	RF 3	5	RF 4
4	d88039629337		online	192.168.225.104	1.023	1.3	0, 00:05:30	N/A	N/A	No	RF 3	5	RF 4
5	d8803961eed1		online	192.168.225.105	1.023	1.3	0, 00:05:30	N/A	N/A	No	RF 3	5	RF 4

Obrázek 4.2: Rozpoznané zařízení RTLS managerem.

Nyní je nutné kotvy mezi sebou vzájemně synchronizovat. To se provádí v sekci Anchor Initialization. Tato inicializace je podrobněji popsána v sekci o implementaci barometrů, jelikož bylo nutné tento postup upravit. Jsou-li kotvy synchronizovány (je zvoleno, která kotva se bude chovat jako master), může být v softwaru Sensmap (původní 2D vizualizační software) nastavena pozice jednotlivých kotev na uživatelem zvoleném plánu budovy.

Pokud vše proběhlo bez problémů, můžeme při pohybu tagu po lokalizovaném prostoru, po přihlášení se k odběru zpráv z WebSocketů, zaznamenat proud dat viditelný na obrázku 4.3. Tyto data budou sloužit klientské 3D aplikaci k aktualizaci modelů jednotlivých komponent systému.


```
▼{body: {id: "22",…}, resource: "/feeds/22"}
  ▼body: {id: "22",…}
    ▼datastreams: [{id: "posX", current_value: "17.131171795074", at: "2018-05-06 17:45:55.879143"},…]
      ▶0: {id: "posX", current_value: "17.131171795074", at: "2018-05-06 17:45:55.879143"}
      ▶1: {id: "posY", current_value: "14.678818863165", at: "2018-05-06 17:45:55.879143"}
      ▶2: {id: "posZ", current_value: "1", at: "2018-05-06 17:45:55.879143"}
      ▶3: {id: "batLevel", current_value: "90%", at: "2018-05-06 17:45:55.879143"}
      id: "22"
    ▼zones: [{id: "5", type: "info", name: "Side Corridor", building_reference: "21", plan_name: "storage",…}]
      ▶0: {id: "5", type: "info", name: "Side Corridor", building_reference: "21", plan_name: "storage",…}
    resource: "/feeds/22"
```

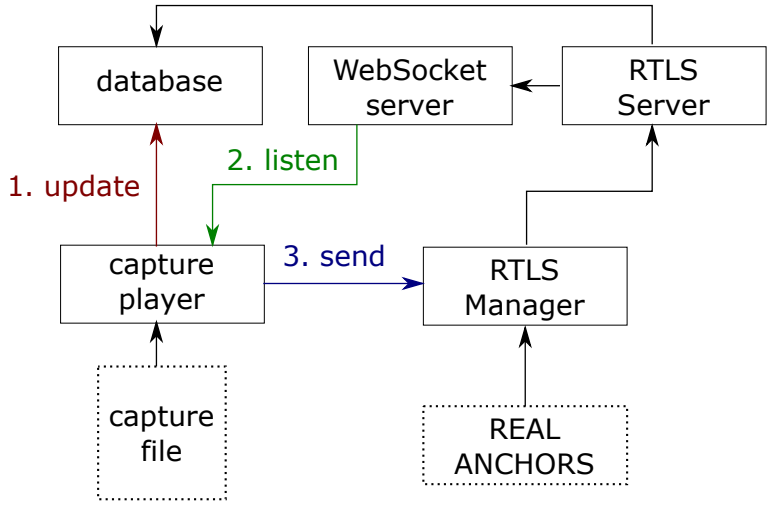

Obrázek 4.3: Data přijímaná z WebSocketů po přihlášení k odběru některého zařízení.

Systém umožňuje kompletní komunikaci mezi serverem a kotvami zaznamenat, je tedy možné si svým způsobem nahrát proces pohybu zařízení pro pozdější přehrávání. Za účelem tohoto přehrávání jsem vytvořil aplikaci Capture Player. Aplikace značně usnadňuje testování systému. Není nutné vždy provádět stejné úkony (přesun tagu do počáteční polohy, spuštění testovaného programu, přesun tagu do cílové polohy), stačí pouze spustit záznam pomocí Capture Playeru. Tento program našel využití také v oblasti technické podpory, kdy si zaměstnanec může přehrát průběh komunikace zařízení, který mu zákazník pořídil a odeslal, a zjistit tak příčinu problému, který se u zákazníka vyskytuje.

4.1.1 Capture Player

Podpůrná aplikace Capture Player je psána v JavaScriptu s využitím frameworku Node.js. Node.js umožňuje vytvořit aplikaci, která pomocí knihovny Express poskytuje uživatelům, kteří se připojí ke koncovému bodu aplikace skrze webový prohlížeč, webovou stránku. Tato webová stránka může dále se serverem komunikovat například pomocí WebSocketů.

K tomu, aby serverová část Capture Playeru správně fungovala, je potřeba mít na serverovém počítači nainstalovanou aplikaci TShark sloužící k práci se zaznamenanými pakety. Tyto pakety jsou Capture Playerem přečteny, analyzovány a následně zasílány na porty, na kterých server standardně přijímá komunikaci od okolních zařízení [15]. Komunikace je ilustrována obrázkem 4.4. Výstupem Capture Playeru je záznam poloh jednotlivých zařízení uvnitř systému v průběhu času.



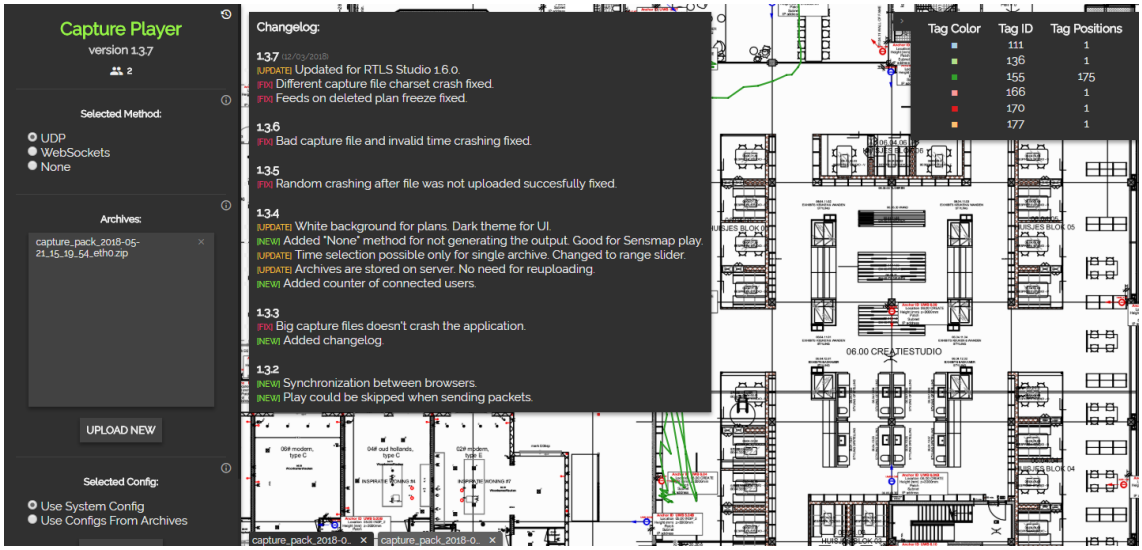
Obrázek 4.4: Komunikace Capture Playeru s ostatními prvky systému. Černé šipky značí standardní průběh komunikace.

Výsledkem je možnost přehrávat komunikaci tak, jak byla zaznamenána. Prostředí aplikace Capture Player je viditelné na obrázku 4.5.

4.2 Klientská aplikace pro 3D vizualizaci

Aplikace pro 3D vizualizaci, dále označována jako Sensmap 3D, je klientská aplikace dostupná skrze portál RTLS Studio, konkrétně její koncový bod `http://server-ip/sensmap3d`. Všechny potřebné závislosti, včetně kódu aplikace, jsou přenášeny jako standardní webová stránka.

Po načtení stránky je zkontrolována dostupnost serveru obsluhujícího spojení pomocí WebSocketů. V případě výpadku je uživatel upozorněn a aplikace se pokouší o znovunavázání spojení. Pokud je úspěšně navázáno spojení se serverem, vznikne hierarchická struktura objektů reprezentující reálná zařízení a prostory, ve kterých se tyto zařízení pohybují. Data o této hierarchii jsou získána dotazem na PHP server, který komunikuje s databází obsahující všechny relevantní informace. Struktura kopíruje předchozí návrh aplikace. Existuje tedy model celého světa, ve kterém se nacházejí budovy s plány, které obsahují lokalizační zařízení (kotvy a tagy). Dále bylo potřeba vytvořit model pro abstraktní objekty, jako jsou historické polohy tagů a zóny.



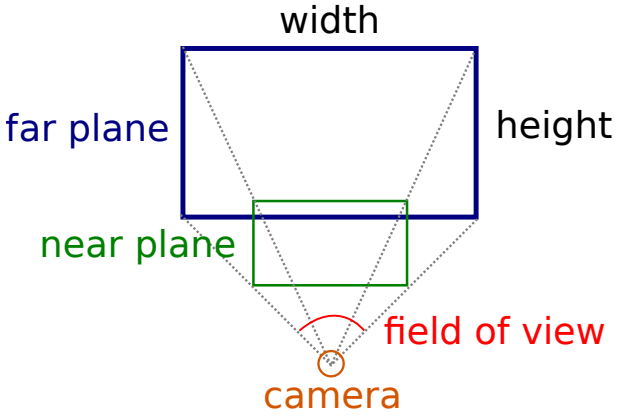
Obrázek 4.5: Ukázka prostředí aplikace pro přehrávání záznamů lokalizačního systému.

Komunikace jednotlivých modelů s okolním světem je zajištěna pomocí třídy *DataLoader*. Tato třída je služba, poskytující data z WebSocketů, na které je aplikace přihlášena (pro každé zařízení jedno spojení). Z dat, dříve demonstrovaných obrázkem 4.3, je patrné, že je potřeba prozkoumat všechny datastreamy konkrétního odebíraného feedu (budova, tag, kotva) a aktualizovat podle typu datastreamu odpovídající hodnoty v modelu daného objektu.

4.2.1 Vytvoření scény ve Three.js

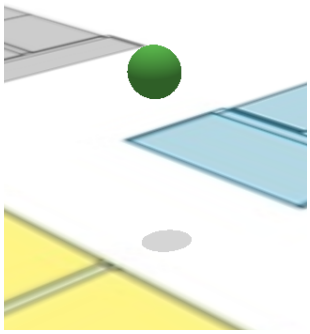
Abych byl schopen v aplikaci vizualizovat jakákoliv data, je potřeba nejprve připravit scénu a všechny její závislosti. Knihovna Three.js má připravené objekty pro základní strukturu své scény. Nejdříve je nutné vytvořit renderer, do kterého se bude veškerá grafika vizualizovat. Tento renderer je postaven na technologii WebGL. Vytvořením rendereru je obdržen grafický kontext aplikace jako DOM element HTML struktury. Element je dále třeba přiřadit jako potomka některého jiného elementu, který se na stránce již nachází (například tělu HTML stránky).

Další potřebné objekty jsou scéna samotná a kamera, jejímž prostřednictvím je na scénu nahlíženo. Scéna představuje objekt, který je kořenem hierarchické struktury všech vizualizovaných objektů. Všechny objekty této struktury jsou potomkem objektu *Object3D*, který je základní vizualizovatelnou jednotkou. U kamery, jak už to standardně bývá ve všech 3D aplikacích, musíme nastavit sadu parametrů. Jedná se o dvě roviny, která omezují renderovaný prostor (definovány dvěma čísly nesoucími hodnotu vzdálenosti roviny od kamery), o zorné pole kamery a o poměr velikostí zobrazovací plochy. Roviny omezující prostor nazýváme *near plane* a *far plane*. Parametry jsou ilustrovány obrázkem 4.6. Z těchto parametrů je následně vytvořena view matice, s jejíž pomocí je počítána projekce trojrozměrného světa na 2D obrazovku. Je potřeba si dát pozor na nastavení příliš vysoké hodnoty near plane. V takovém případě neuvidíme objekty, které jsou mezi touto rovinou a kamerou. Naopak také nebudou vidět objekty, které leží až za rovinou far plane. Tímto omezením renderovaného prostoru zmenšujeme nároky na výpočetní sílu grafické karty počítače [4].



Obrázek 4.6: Parametry kamery pro 3D vizualizaci.

V neposlední řadě je třeba scénu osvětlit. Pro udržení jisté míry viditelnosti i v zastíněných oblastech používám ambientní osvětlení. Druhým zdrojem světla je směrové světlo přicházející svrchu. Touto kombinací světel docílím 3D efektu na vizualizovaných objektech. S pozdějším zavedením barometrů pro určování výšky tagů se může stát, že není pro uživatele na první pohled patrné na jakém místě plánu se tag přesně nachází. Situaci demonstruje obrázek 4.7. Řešením tohoto problému je umístit na plán pod lokalizovaný objekt stín. Tímto je docíleno toho, že ať je objekt v jakékoliv výšce, vždy známe jeho přesnou polohu vůči plánu. Počítání vržených stínů pro každý objekt scény je poměrně náročná operace. Mnohem méně výpočetně náročně je přidat do scény objekt nový. Pod každým tagem je proto vytvořen válec, který má stejný průměr jako koule reprezentující tag. Tento válec tedy představuje stín objektu a je umístěn do téměř nulové výšky. Výška objektu není přesně nulová, protože by vizualizace v místě protnutí válce s plánem mohla problikávat. Tento jev se objevuje při přepočítávání vzdálenosti objektu od kamery ve vertex shaderu a při chybách necelých čísel se pro různé pixely výsledného obrazu může stěna objektu střídat se stěnou objektu nacházejícím se na stejném místě [20]. Tento vizuální jev může kazit dojem výsledné vizualizace, je mu proto třeba předcházet.



Obrázek 4.7: Ukázka neurčitosti polohy při nenulové výšce objektu.

Nakonec je vhodné vytvořit pro uživatele nějakou formu ovládání celé scény. U knihovny Three.js se ovládání označuje standardně *Controls* a existuje několik připravených variant. Já jsem zvolil variantu *OrbitControls*, jelikož je nejvhodnější pro zkoumání nějakého konkrétního tělesa (například plán budovy). Nebyl jsem však spokojen s tím, jakým způsobem jsou v knihovně obsloužena vstupní zařízení, proto jsem tuto část ovládání přepsal tak,

aby ovládání myší nebylo omezeno pouze na výsek oblasti okna, ale aby naopak při uživatelské akci byla myš uzamčena do následné akce uvolnění (při otáčení kamery se ukazatel uzamkne, je tedy možné rotovat o libovolný úhel a ukazatel je odemčen až po uvolnění tlačítka myši).

Vše popsané v této sekci se odehrává ve třídě *RenderingContext*, která svým způsobem zaobaluje základní vlastnosti knihovny Three.js.

4.2.2 Hierarchie objektů ve scéně

Jak již bylo nastíněno v kapitole návrhu aplikace, každý objekt scény je v mé implementaci možné rozdělit na dvě části: Model a Mediátor. Model v tomto případě dědí ze třídy *Observervable*. Observervable je v Javascriptu vytvořen následujícím způsobem:

- nejprve je v konstruktoru vytvořena mapa[1] všech observerů, tedy callbacků přihlášených k odběru konkrétních akcí;
- dále je přítomna funkce pro přihlášení nového observera, která pro existující akci přidá callback do pole u položky mapy odpovídající akci, nebo pro neexistující akci vytvoří v mapě nový záznam;
- nakonec je definována funkce, pomocí které může observable vyvolat všechny callbacky přihlášené ke konkrétní akci;

Máme-li připraven takovýto model (může se jednat například o model tagu), můžeme ho přidat do hierarchie modelů a automaticky pomocí továrny k němu zvolit odpovídající Mediátor. Mediátor má v sobě zahrnut odkaz na Model, pro který byl vytvořen, není tedy problém se přihlásit k odběru všech událostí na které chceme vizuálně reagovat. Po přihlášení k odběru již stačí v mediátoru vytvořit skutečný vizualizovaný objekt knihovny Three.js a nastavit podle modelu všechny parametry a funkce, které mají být volány, pokud dojde k některé akci (například změna konkrétního parametru). Každý Mediátor má navíc v sobě funkci, která je volána pro každý renderovaný snímek. Tímto způsobem je možné ovládat objekt v průběhu vykreslování.

Abychom byli schopni vytvářet strukturu objektů automaticky, musí navíc každý Mediátor obsahovat funkce pro přidání potomka, které jsou volány Modelem v případě, že je mu některý potomek skutečně přidán. Pro tyto účely je vytvořen obecný Mediátor, ze kterého ostatní Mediátory dědí.

4.2.3 Uživatelské rozhraní

Uživatelské rozhraní aplikace je složeno ze tří hlavních komponent. První komponentou je horní panel, sloužící k výběru plánu a informování uživatele o tom, o jaký software se jedná a jaká je spuštěná verze tohoto softwaru. Druhou komponentou je menu vytvořené s pomocí knihovny dat.gui upravené pro potřeby mojí aplikace. Poslední a také nejdůležitější je část, do které je vykreslována samotná vizualizace.

Horní panel je vytvořen čistě jako element HTML stránky. Pro vytvoření struktury tlačítek a jejich stylu jsem využil knihovnu Bootstrap[2]. Tlačítka plánů v panelu jsou rozdělena podle toho, ve které budově se plán nachází. Pro budoucí potřeby bude pravděpodobně horní

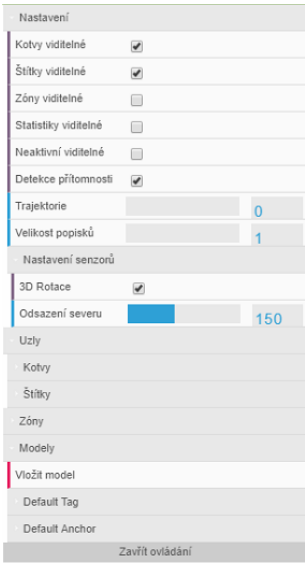
[1] `https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Map`
[2] `https://getbootstrap.com/`

panel rozšířen o ovládání spjaté s uživatelskými účty a konkrétně přihlášenými uživateli. Pro demonstraci je přiložen obrázek 4.8. Vytváření seznamu tlačítek probíhá dynamicky s využitím jQuery na základě získaných parametrů z RESTu.

Obrázek 4.8: Výřez horního panelu aplikace Sensmap 3D.

Boční menu slouží převážně k ovládání celé vizualizace. Obrázek 4.9 ukazuje jak menu ve výsledku vypadá. Menu jsem rozdělil na čtyři části:

- nastavení - nastavení vizualizace;
- uzly - výčet připojených zařízení;
- zóny - výčet vytvořených zón;
- modely - správa 3D modelů;



Obrázek 4.9: Ukázka menu aplikace.

Většina položek menu je určena ke skrývání a zobrazování různých typů objektů ve scéně. Jelikož je GUI aplikace napojeno pouze na třídu *MainView*, je nutné skrze *WorldController* iterovat všemi budovami a plány, abychom byli schopni schovat všechny objekty daného typu. Tímto postupem se dostáváme až k modelům konkrétních objektů a změnou jejich stavu vyvoláme také změnu vizualizace.

Výjimkou je možnost schovat neaktivní tagy. Zde nestačí pouze aktualizovat model, nýbrž je potřeba držet informaci o čase poslední změny polohy. Pokud doba mezi časem

poslední polohy a současností převyšuje předem navolenou konstantu, můžeme považovat tag za neaktivní a schovat jej.

Menu také umožňuje zahájit sledování některého z tagů. To je prováděno prostřednictvím knihovny Three.js. Pokud chceme, aby kamera sledovala některý objekt, stačí přidat kameru mezi potomky tohoto objektu ve scéně. Knihovna si sama poradí s odebráním kamery z původního místa v hierarchii scény a umístěním na místo nové.

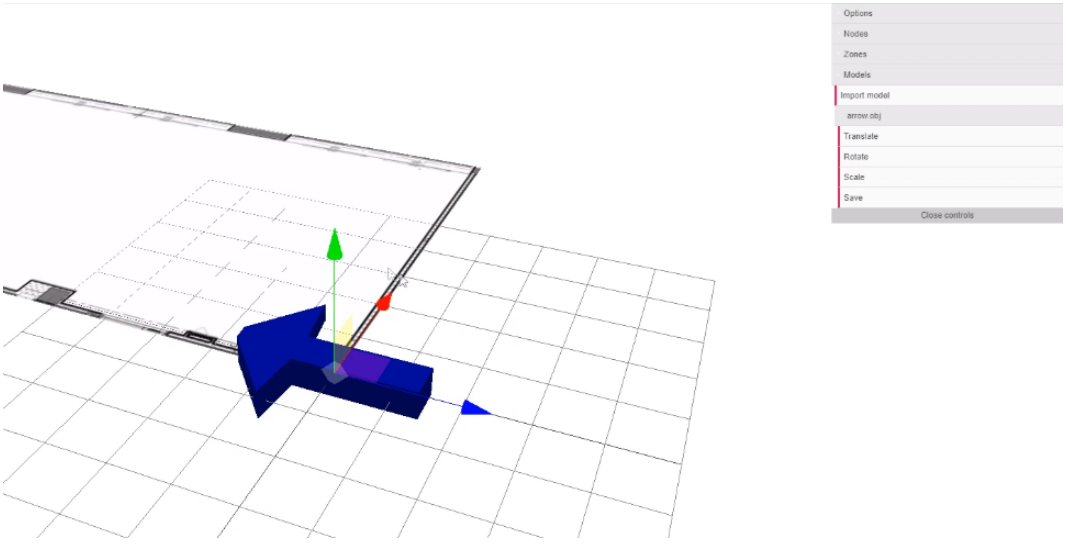
Editor modelů

Modely nahrané uživatelem nemusejí vždy odpovídat měřítku plánu, ve kterém mají být vizualizovány. Jedním řešením takového problému je model vždy upravit v externím programu před nahráním do vizualizační aplikace. Taková operace je však zbytečně namáhavá a ne vždy má uživatel připraveny potřebné nástroje k tomu, aby ji mohl provést. Pro tyto případy jsem do aplikace přidal jednoduchý editor importovaných modelů.

Všechny modely jsou vypsány ve spodní části menu aplikace. Rozbalením jejich složky a kliknutím na položku Edit je uživatel přenesen do editoru, jehož prostředí demonstruje obrázek 4.10. Všechny prvky aplikace jsou schovány a pro účely editoru je vytvořen nový model s mediátorem, který neodpovídá žádnému reálnému zařízení. Uživateli jsou umožněny 3 operace manipulující vertexy modelu:

- translate - pro posun středu modelu (vždy odpovídá rohu plánu);
- rotate - pro změnu natočení modelu;
- scale - pro změnu měřítka modelu;

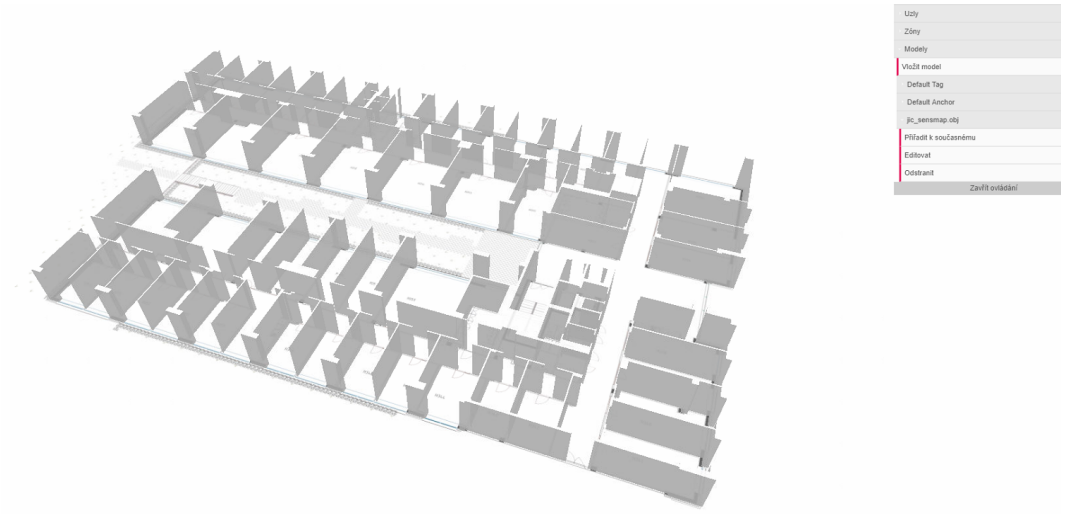
Modely mohou být také nahrány jako součást plánu pro zkvalitnění vizualizace. Plán s modelem je zobrazen na obrázku 4.11.



Obrázek 4.10: Editor modelů vizualizační aplikace.

4.2.4 Ukládání parametrů nastavení

Webová aplikace umožňuje pomocí uživatelského rozhraní měnit řadu nastavení, je proto žádoucí, aby tato nastavení nepozbyla platnost v případě nového načtení stránky. Existuje



Obrázek 4.11: Plán po aplikaci 3D modelu.

několik způsobů, jak toho docílit v prostředí JavaScriptu s API backendem na serveru. Nejjednodušším řešením, které bylo také v aplikaci implementováno, je využít takzvaný local storage webového prohlížeče. Tato nastavení jsou uchovávána lokálně na počítači, na kterém byla spuštěna aplikace. V budoucnu bude pravděpodobně s příchodem uživatelských účtů potřeba nastavení uložit na server, aby mohli uživatelé přenášet nastavení mezi počítači v závislosti na tom, na kterém účtu jsou přihlášeni.

Local storage[3] představuje pole dvojic sestávající z unikátního klíče a hodnoty tomuto klíči přiřazené. Toto nastavení je možné získat bez závislosti na konkrétním sezení. Pro jednotnost před klíčem vždy uvádím prefix sewio-sensmap3d, kterým označuji, že se jedná o data pro aplikaci Sensmap 3D společnosti Sewio.

Při načítání klienta se načtou také hodnoty z local storage, a pokud jsou nastaveny některé konkrétní parametry, využívám jejich hodnot pro inicializaci nastavení. V některých případech stačí pouze uložit data do proměnné a později je využít. V jiných případech je třeba data ještě dále aplikovat na modely v aplikaci, a to až po jejich úspěšném načtení.

Mimo jiné v mé aplikaci existuje ještě jedna forma vynucení nastavení. Toto vynucení se provádí skrze parametry obsažené v URL aplikace. Změna počáteční konfigurace nastavení pomocí URL je dostupná kvůli některým konkrétním případům užití aplikace. Prvním případem je demonstrační aplikace, ve které chceme mít definovány veškeré nastavení pro uživatele již při jejich prvním spuštění stránky. Dalším případem je server s aplikací dostupnou na jiné IP adrese. V tomto případě se pro komunikaci využije adresa poskytnutá jako parametr. Toto nastavení bývá do URL zahrnuto při spouštění aplikace skrze portál RTLS Studio. Posledním případem využití může být předávání nastavení mezi uživateli. V nastavení se dá přenášet také pozice kamery, tím pádem si uživatelé mohou předávat odkaz s pozicí kamery zaměřenou na konkrétní místo.

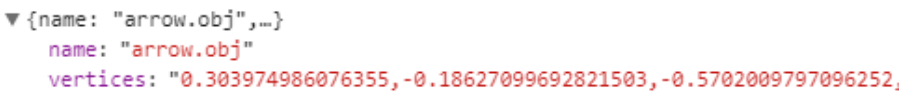
4.2.5 Vkládání 3D modelů

Aplikace Sensmap 3D je, jak z názvu vyplývá, aplikace zobrazující svět ve třech rozměrech, bylo by tedy zvláštní kdyby takováto vizualizace neumožňovala import vlastních 3D modelů.

[3] `https://developer.mozilla.org/en-US/docs/Web/API/Storage/LocalStorage`

Existuje mnoho formátů, ve kterých jsou 3D modely uschovávány, já však pro moji aplikaci zvolil formát OBJ. Je to jeden z nejzákladnějších formátů, má jednoduchou strukturu a není problém model z jakéhokoliv jiného formátu do OBJ převést [14].

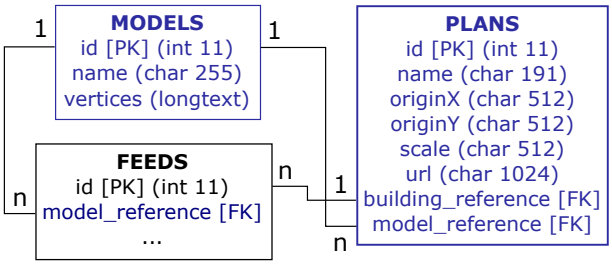
Nahrávání modelu do aplikace je umožněno skrze standardní JavaScriptový File API[4] rozšířený o funkce jQuery. Pokud uživatel zvolí soubor, klient zaznamená změnu a pokusí se ze souboru extrahovat geometrii 3D modelu. Pokud je soubor v pořádku a podaří se úspěšně načíst geometrie modelu, je tato geometrie uložena mezi ostatní uživatelské geometrie. Nakonec stačí pouze pro jednotlivé objekty zvolit načtený 3D model, čímž je jeho původní geometrie nahrazena geometrií načteného modelu. Jistě ale každého napadne, že pokud je aplikován tento postup, při dalším načtení stránky budou všechny uživatelské modely ztraceny a proces bude nutné znovu opakovat. Aby k tomu nedocházelo, je potřeba model uložit na server. Již jsem se v kapitole o návrhu aplikace zmiňoval o tom, že model bude uložen do databáze jako standardní řetězec. Je tedy potřeba upravit databázi serveru společnosti Sewio. Dále je potřeba upravit API tak, aby model od klienta přijímalo. Jakmile jsou tyto úpravy zavedeny, může být model odeslán ve formátu JSON, jako ilustruje obrázek 4.12.



Obrázek 4.12: Výřez z JSON komunikace mezi klientem a serverem pro uložení nového modelu do databáze.

Úprava databáze

Server společnosti Sewio je podstaven nad MySQL databází. V databázi jsou uložena data o všech zařízeních, která se v systému vyskytují, o jejich současných stavech, a pokud má uživatel nastaveno ukládání informací do databáze, nachází se zde také data o stavech předchozích. Obrázek 4.13 ukazuje části databáze, které byly upraveny. Černá barva v obrázku značí části původní a modrá části které byly přidány. V obrázku nejsou zahrnuty změny, které byly provedeny později pro použití s barometrickými senzory.



Obrázek 4.13: Upravené tabulky databáze pro ukládání modelů.

V původním návrhu databáze byly plány součástí textového popisku jednotlivých feedů (společná tabulka pro budovy a všechny zařízení). Oddělení plánů do samostatné tabulky umožňuje snadnější manipulaci a data jsou lépe pochopitelná. Veškeré úpravy databáze musí být zavedeny také do instalačního skriptu softwarového balíčku RTLS studio.

[4] https://developer.mozilla.org/en-US/docs/Web/API/File/Using_files_from_web_applications

Úprava API

API je obsluhováno backendem vytvořeným v PHP s využitím frameworku Slim[5]. Na všechny základní RESTové dotazy tento PHP server odpovídá. Pro uložení modelu skrze API je nutné přidat nové koncové body, a rozšířit tím PHP server [13]. Koncové body jsou popsány tabulkou 4.1.

GET	/models	Vrací všechny 3D modely uložené v databázi.
POST	/models	Vytváří nový 3D model v databázi podle poskytnutých parametrů.
PUT	/models/[id]	Aktualizuje 3D model s odpovídajícím ID podle poskytnutých parametrů.
DELETE	/models/[id]	Smaže 3D model s odpovídajícím ID.

Tabulka 4.1: Rozšíření API pro ukládání 3D modelů.

Framework Slim je velice intuitivní a přidávání nových koncových bodů je relativně snadné. Stačí pouze definovat řetězcem, jak má koncový bod vypadat, jaké má mít proměnné a přiřadit k němu funkci, která má být v případě dotazu na koncový bod provedena. Takováto funkce se skládá ze tří částí:

- kontrola API klíče uživatele;
- dotaz do databáze vykonávající požadovanou akci;
- odeslání odpovědi klientovi, který akci vyžadoval;

Dotazy do databáze jsou v PHP prováděny skrze rozšíření PHP Data Objects[6]. Je důležité všechny parametry dotazu do databáze přikládat skrze rozhraní *bindParam*, aby nebylo možné databázi napadnout útokem SQL injection.

Kromě koncových bodů týkajících se modelů bylo potřeba také upravit koncové body pro jednotlivé plány. To vyplývá z úprav, které byly provedeny na databázi v předchozí části textu. Nahrazené koncové body demonstruje tabulka 4.2. Úpravy byly provedeny především pro jednodušší pochopitelnost API.

POST	/buildings/[id]/plans	ZACHOVÁNO
GET	/buildings/[id]/plans	ZACHOVÁNO
PUT	/buildings/[id]/plans/[name]	/plans/[id]
DELETE	/buildings/[id]/plans/[name]	/plans/[id]

Tabulka 4.2: Nahrazení koncových bodů pro práci s plány.

4.2.6 Rotace zařízení

Stejně jako souřadnice na kterých se má zařízení nacházet, jsou ze serveru přijímány data o rotaci zařízení. Možnost rotovat objekty v prostoru je jednou z výhod trojrozměrné vizualizace. Tato data mají formu čtyř hodnot quaternionu.

[5] `https://www.slimframework.com/`

[6] `http://php.net/manual/en/book.pdo.php`

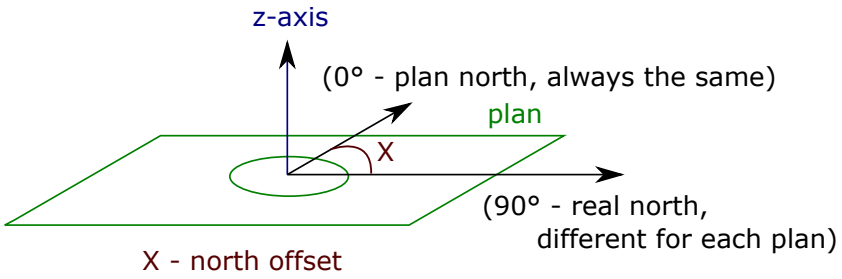
Aby byly hodnoty vždy drženy pohromadě, jsou ze serveru odesílány ve formě řetězce, ve kterém jsou jednotlivé hodnoty odděleny čárkou. Takový řetězec je nejprve nutné zpracovat. Ještě před vytvořením rotační matice je také potřeba hodnoty správně přeskládat v závislosti na tom, jak je na zařízeních kvaternion formulován. Přeskládáním je docíleno vzájemné kompatibility mezi pojetím souřadného systému ze strany klientské aplikace a ze strany jednotlivých zařízeních. Převod mezi těmito systémy je následující:

$$q_{dev} = (x, y, z, w) \tag{4.1}$$

$$q_{app} = (-z, -x, -y, w) \tag{4.2}$$

V hierarchii scény je třeba dávat pozor na to, s jakým objektem je rotováno. Pokud bych rotoval objekt, který má ve scéně další potomky, jejich souřadný systém by byl pootočen. Z tohoto důvodu pokud rotuji některá zařízení, vždy rotuji pouze mesh, který drží přímo geometrii zařízení. Důležité je také podotknout, že geometrií zařízení rotuji pouze v případě, že edituji model samotný. Pokud bych provedl rotaci geometrie, tak by se mi místo konkrétního zařízení natočily všechny objekty geometrii využívající.

Jelikož budovy nejsou vždy orientovány směrem k severu a plány také ne, je třeba v uživatelském rozhraní umožnit uživatelům nastavit hodnotu pomyslného severu ve stupních. Tato hodnota hraje významnou roli při rotování tagu, určuje totiž, kde se nachází pomyslných nula stupňů rotace kolem vertikální osy. Hodnotu pomyslného severu neboli *north offset* vysvětluje obrázek 4.14. Máme-li objekt, který směřuje na sever (nula stupňů), musíme k jeho rotaci přičíst *north offset*, aby tento objekt směřoval ke skutečnému severu, místo severu našeho plánu.Vyskytuje se zde však jedna potíž, a to že rotaci tagu držíme v quaternionech, kdežto posun magnetického severu ve stupních, jako Eulerův úhel. Pro každou novou rotaci je potřeba nejprve Eulerův úhel převést na quaternion. Pro výslednou rotaci tedy postačí quaternion rotace tagu vynásobit s quaternionem vzniklým z Eulerova úhlu. Touto operací dostáváme quaternion, který je rotovaný vzhledem k magnetickému severu určenému uživatelem. Výsledný quaternion je kompatibilní s naší vizualizací, můžeme ho proto přímo využít.



Obrázek 4.14: Ukázka odsazení severu na všech plánech.

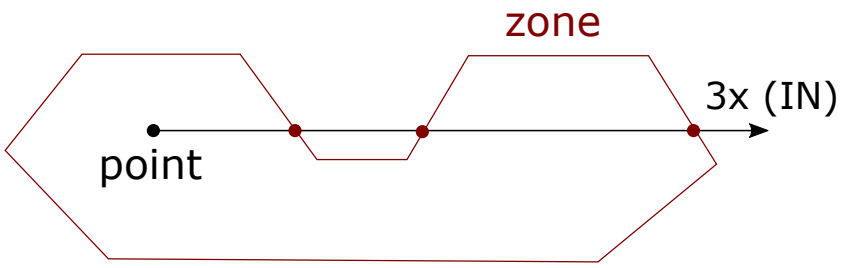
4.2.7 Podpůrné výpočty

Chod aplikace pro vizualizaci 3D lokalizace by se neobešel bez pomocných výpočtů, které klient provádí během svého chodu. Nepodstatnější z nich popíši v této podsekci.

Detekce přítomnosti v zóně

Jedná se v podstatě o problém náležitosti bodu do nějakého mnohoúhelníku. Tento problém má mnoho možných řešení. Jedním z nejelegantnějších je řešení algoritmem z roku 1962, který bývá označován jako even-odd rule, nebo ray crossings a je využit například v SVG.

Máme-li bod a chceme zjistit, jestli tento bod leží v námi definovaném mnohoúhelníku, můžeme vrhnout paprsek kterýmkoliv směrem (pro nejjednodušší výpočet bývá využita osa X) a spočítat kolikrát paprsek protal hranu mnohoúhelníku. Z obrázku 4.15 vyplývá, že pokud je počet protnutí lichý, bod leží uvnitř mnohoúhelníku. V opačném případě se bod musí nacházet vně mnohoúhelníku [18].



Obrázek 4.15: Test náležitosti bodu do mnohoúhelníku pomocí metody even-odd rule.

Existují situace, ve kterých může algoritmus mylně rozhodnout o náležitosti bodu do mnohoúhelníku. Jedná se o situace, kdy bod leží poblíž hran mnohoúhelníku nebo v blízkosti některého z jeho vrcholů. Tyto situace jsou však pro mé využití zanedbatelné a jednoduchost a rychlost řešení bez dodatečných podmínek je důležitější, jelikož k výpočtu v některých případech dochází pro každou příchozí polohu.

Výpočet času potřebného pro animaci

Aby přechod mezi dvěma polohami lokalizovaných zařízení nebyl diskrétní tak, jak ho klient přijímá od serveru, ale aby se objekt po scéně pohyboval plynule, je potřeba mezi dvěma polohami interpolovat. K interpolaci mezi dvěma polohami používám malou knihovnu Tween.js[7]. Knihovna po vytvoření animace předává proměnným hodnoty v závislosti na čase, který uběhl od počátku animace. Aby se tyto hodnoty aktualizovaly, je třeba pro každý snímek běhu aplikace volat pomocnou funkci update. Knihovna také umožňuje interpolovat hodnoty po křivce. Každá interpolace nicméně potřebuje vědět, kolik času má trvat přechod z původní pozice na pozici následující. Tato doba je vždy proměnlivá v závislosti na mnoha parametrech (zpoždění na síti, zpoždění výpočtu, vysílací frekvence).

Jako nejlepší řešení se ukázalo využít medián tří posledních rozdílů v časech mezi polohami. Toto řešení je dostatečně jednoduché, tím pádem nevadí, že k výpočtu dochází pro každou příchozí pozici každého zařízení, ale s kombinací s vhodným typem interpolace mezi polohami přináší velice přirozené výsledky. Pokud několikrát po sobě čas trval příliš dlouho, je lepší další čas interpolace nahradit konstantní hodnotou, jinak bude docíleno jevu, že se zařízení pohybuje z jednoho místa na druhé, a to velmi malou rychlostí. Pokud mu je nastavena menší hodnota času, je z pohybu patrné, že se pouze synchronizuje pozice s realitou. Tento jev může být ještě posílen nahrazením klasické lineární interpolace interpolací po křivce, podél které se zařízení pohybuje pomaleji na koncích animace než v jejím středu.

[7] `https://github.com/tweenjs/tween.js/`

Aktualizace historických poloh

Aplikace umožňuje vizualizovat také polohy, které již nejsou aktuální. Uživatel má v takovém případě povědomí o tom, kudy se zařízení pohybovalo. V nastavení může být počet těchto historických poloh omezen, aby nedocházelo k poklesu výkonu aplikace.

Nejlogičtějším řešením, které se nabízí, je při omezeném počtu mazat polohy, které tento počet přesahují a vytvářet vždy nové. To ovšem znamená, že je potřeba vytvářet také vždy nové objekty scény, což není v prostředí knihovny Three.js optimální. Dochází zde k výraznému nárůstu spotřeby paměti, a to i při pravidelném využívání garbage collectoru.

Aby aplikace co nejméně zatěžovala paměť, místo mazání starých poloh a vytváření nových, využívám staré polohy a upravuji jejich parametry, aby odpovídaly polohám novým. Při využití tohoto postupu v paměti pouze neustále rotuje pole stejných objektů ve scéně.

4.3 Zpracování barometrických dat na serveru

Práce s barometrickými daty na serveru pro úspěšný výpočet vertikální pozice lokalizovaných zařízení s sebou nese několik úskalí. Výčet těchto komplikací, které budu dále rozebírat v této sekci je následující:

- jak připravit a uložit data potřebná pro výpočet;
- jak a kdy zařízení vzájemně synchronizovat;
- jak si poradit s chybou měření jednotlivých zařízení;
- jak provádět výpočet;

Jednotlivé položky budou popsány každá ve své vlastní podsekci, ve které problém popíši více detailněji, dále navrhnu řešení a zmíním se o implementaci tohoto řešení.

4.3.1 Příprava a uchování dat

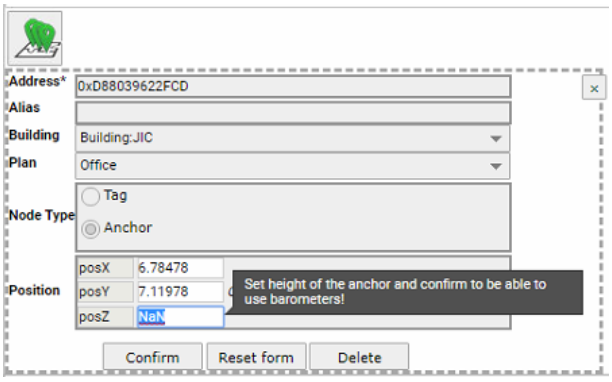
Prvním problémem je příprava a uschování dat. Je třeba se zamyslet nad tím, jaká data potřebuji k výpočtu výšky zařízení, kde tato data obdržím a jakým způsobem je uložím.

Pro ilustraci budu předpokládat, že mám dvě lokalizační buňky, které se nacházejí přímo nad sebou. Každá buňka má své vlastní kotvy pro lokalizaci a společně sdílejí jeden tag, který může mezi těmito buňkami přecházet. Všechny parametry této instalace popisuje obrázek 4.18. Z obrázku je patrné, že pro správný výpočet potřebujeme znát výšku každého plánu vůči plánu referenčnímu (jedná se o plán přízemí, nebo nejnižšího lokalizovaného patra). Dále potřebujeme informaci o tom, v jaké výšce se která kotva nachází vůči podlaze plánu, na kterém je umístěna. Z těchto dvou údajů jsme schopni pro každou kotvu spočítat její výšku (opět v závislosti vůči nejníže položenému plánu). Dalším důležitým parametrem je informace o tom, do jaké výšky lokalizace plánu zasahuje. Aby bylo dosaženo nejlepších výsledků, je nutné instalaci navrhnout tak, aby se lokalizované buňky nepřekrývaly.

Zadávání hodnot

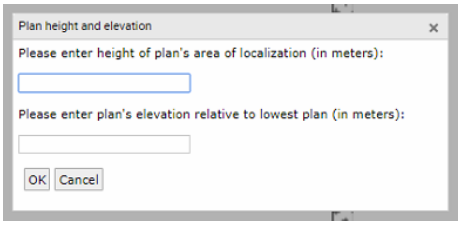
Všechny parametry musí uživatel sám poskytnout serveru. Toto je umožněno skrze původní vizualizační aplikaci, kde je uživatel nově upozorněn při rozmísťování kotev, které nemají nastavenou výšku, že je tento parametr doposud nedostupný, a uživatel by měl tuto informaci zadat na příslušné místo v aplikaci, jako na obrázku 4.16. Tato funkcionalita byla

implementována do původní vizualizační aplikace, protože ta stále plní funkci konfiguračního nástroje pro celý systém. Nová aplikace má spíše demonstrační charakter a je vhodná pro použití s nastaveným systémem.



Obrázek 4.16: Upozornění na chybějící parametr kotvy.

Výška lokalizované buňky a výška plánu jsou zadávány v části aplikace pro správu plánů. Na obrázku 4.17 je viditelné modální okno pro zadávání těchto dvou parametrů.



Obrázek 4.17: Modální okno pro zadávání výšky buňky a výšky plánu.

API a databáze

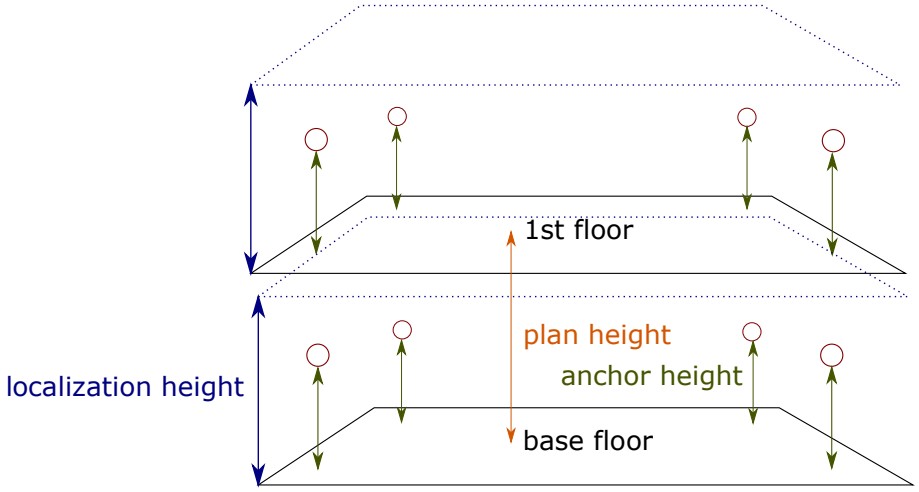
K aktualizaci hodnot kotev slouží dříve dostupný koncový bod *feeds*, který umožňuje upravovat parametry jednotlivých feedů (zařízení, budovy). Pro plány naopak existuje koncový bod plans definovaný v předchozí části textu, včetně databázové tabulky plans. Do této tabulky nově přibyly dvě hodnoty elevation a height, které odpovídají hodnotám plan height a localization height z obrázku 4.18. Obě hodnoty jsou typu double a jejich hodnota je určována v metrech.

4.3.2 Inicializace zařízení a chyba měření

Jestliže jsou všechny hodnoty nastaveny, je nutné provést inicializaci kotev skrze RTLS Manager. Při této inicializaci jsou analyzovány naměřené hodnoty barometrů všech kotev v systému a s přístupem k informacím o jejich skutečné výšce může být měření přepočítáno.

Žádný senzor bohužel není dokonalý, a proto je také potřeba počítat s chybou měření. Tato chyba může být různě veliká pro každé zařízení v systému, zpravidla však zůstává stále stejná, jde tedy spíš o posun hodnot, který se může pohybovat v rozmezí od nuly až do několika desítek Pascalů. Tento problém je také řešen během inicializace kotev.

Nejprve je zvolena jedna referenční kotva. Tlak naměřený na této kotvě budeme považovat za správný. Máme tedy kotvu s následujícími parametry:



Obrázek 4.18: Demonstrační instalace dvou buněk a jejich parametrů.

$$A = (pressure, height_a, height_p) \tag{4.3}$$

Absolutní výšku kotvy získáme sečtením hodnot $height_a$ a $height_p$. Pro každou další kotvu porovnáme tlak vypočtený na základě rozdílu absolutní výšky této kotvy a kotvy referenční s tlakem naměřeným na kotvě. Vzorec tohoto výpočtu můžeme odvodit z 2.8 následovně:

$$p_{next} = \frac{p_{ref}(16000(1 + 0.004t_m) - \Delta z)}{16000(1 + 0.004t_m) + \Delta z} \tag{4.4}$$

Porovnáním získáme chybu měření této kotvy, kterou je možné uložit do databáze pro budoucí výpočty.

Dále potřebujeme zkalibrovat tag, který má sloužit k lokalizaci. Kalibrace je také prováděna skrze RTLS Manager. Pro zjištění chyby měření barometru na tagu musíme položit tag do nějaké konkrétní výšky v prostředí některé lokalizační buňky. Není zde nutná vzájemná komunikace s tagem, stačí pouze po dobu, kdy tag leží ve známé výšce, odebírat zprávy o stavu jeho barometru. Chyba je spočtena stejně jako u kotev a následně také uložena. Prostředí pro kalibraci tagu je vizualizováno obrázkem 4.19.

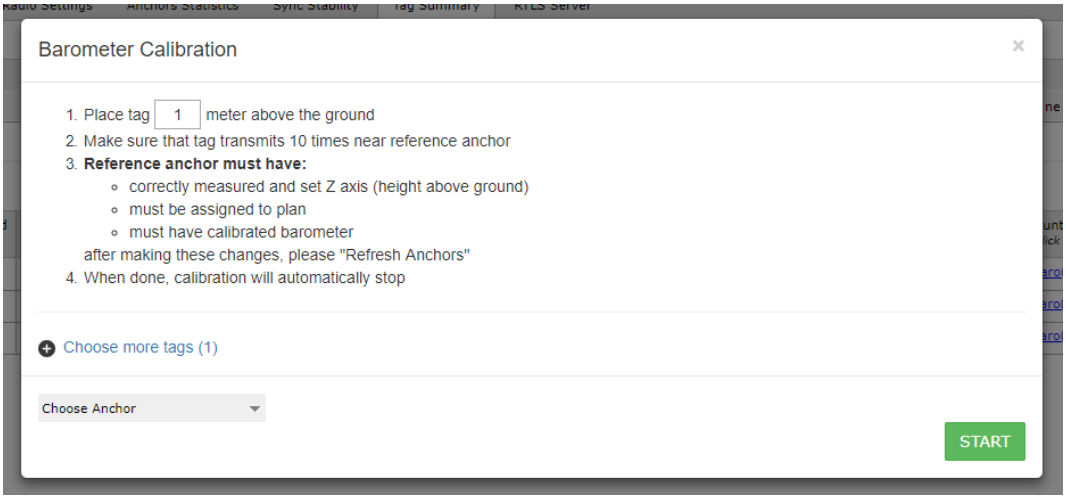
V databázi jsou pro tyto informace vytvořeny dvě nové tabulky. Strukturu tabulek demonstruje obrázek 4.20. Kotvy zůstávají stále na stejném místě, můžeme proto tedy uložit do databáze pouze konkrétní hodnotu tlaku, teploty a výšky, která byla definována. Tagy naopak svoji pozici neustále mění, proto je do databáze uložena pouze informace o chybě proti referenční kotvě.

4.3.3 Výpočet

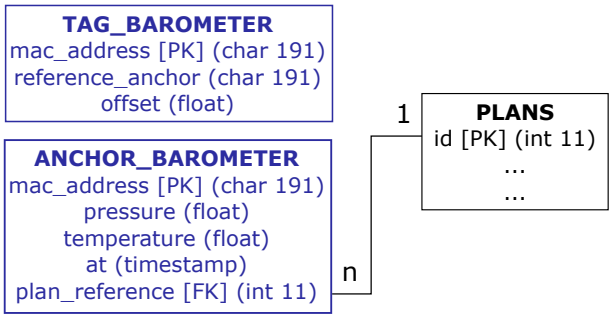
Abych mohl implementovat samotný výpočet výšky pomocí barometrů, musím nejprve analyzovat jakým způsobem proces pro výpočet lokalizace na serveru funguje.

Analýza serveru

Server můžeme rozdělit na několik částí, které se starají o všechny části výpočetního řetězce. Pro moji práci jsou nejdůležitější:



Obrázek 4.19: Modální okno pro kalibraci barometrů v prostředí RTLS Manageru.



Obrázek 4.20: Tabulky pro ukládání hodnot barometrů.

- příjem dat ze zařízení;
- práce s WebSockety;
- selekce kotev;
- výpočet multilaterace;

Tyto části většinou pracují paralelně a nezávisle na sobě (kromě situací, kdy jedna část potřebuje data od jiné). Segment pro příjem dat ze zařízení má dvě hlavní funkce. Přijímá veškerá data, která jsou posílána na server a převádí je do serverem zpracovatelné formy. Datové struktury, které jsou při tomto procesu vytvářeny, jsou v textu popsány později. Úsek musí být obohacen o schopnost zpracování barometrických dat ze zařízení.

Pro zpracování WebSocketů je přítomna další část. Jak již popis napovídá, tato část přijímá data z WebSocket serveru. Data obsahují převážně informace o tom jaké uživatel provedl změny na pozici kotev systému. Šíření těchto dat má na starosti PHP server reagující na volání některých koncových bodů. Sektor musí být rozšířen o možnosti zpracování nastavení výšky kotev pro budoucí inicializaci.

Selekce kotev je další důležitý díl serveru. Tato část na základě dostupných dat volí, které kotvy mají být využity pro konkrétní výpočet. Zde je nutné doplnit funkce, které chytře rozlišují, na kterém poschodí se tag nachází. To vše za pomocí dat z barometrů,

které jsme v předchozích částech zpracovali. Pomocí volby poschodí tak můžeme snadno vyfiltrovat kotvy, které na tomto poschodí nemají být dostupné.

Nakonec musí být zasaženo do samotného výpočtu multilaterace. Zde budou doplněny funkce, které pro konkrétní vodorovnou pozici tagu vypočtou její vertikální pozici z dostupných barometrických dat. Dále jsou tyto informace využity k optimalizaci multilaterace samotné na základě informací o tom, v jaké výšce k výpočtu došlo. Tímto je tedy vylepšena 2D projekce vypočtené polohy.

Datové struktury

Pro práci s daty je na serveru připraveno několik datových struktur. Tyto struktury slouží k uložení parametrů jednotlivých zařízení a také převedení do formátů, které mohou být zpracovány ostatními komponentami systému.

Kotvy a tagy jsou uloženy ve strukturách *TAnchorHashList*, respektive *TTagHashList*. Tato struktura obsahuje informaci o zamčení těchto zařízení a také hashovací tabulky *GHashTable* knihovny Glib[8], pro rychlé vyhledávání. V tabulkách jsou obsažena všechna zařízení v systému. Pro každé konkrétní zařízení je dále dostupná struktura *TTag* a *TAnchor*. V těchto strukturách se již nacházejí konkrétní data daného zařízení. Obě struktury musejí být rozšířeny o barometrická data.

Dále na serveru bývá využívána datová struktura *cJSON*, která slouží k práci s daty parsovanými právě z formátu JSON, který je přijímán z WebSocketů. Tato struktura je obsažena v knihovně se stejným názvem cJSON[9].

Proces výpočtu

Hlavním cílem je obohatit proces výpočtu o vertikální lokalizaci a tato data uložit. V příštích krocích také rozhodnout, které kotvy mají být pro výpočet využity (podle patra ve kterém se tag nachází), spočíst novou vertikální pozici a všechna data odeslat.

Samotný výpočet probíhá ve vláknu *multilateration*. V tomto vláknu se nachází smyčka, která výpočet provádí vždy, když se v asynchronní frontě vyskytuje struktura odpovídající multilateraci obsahující dostatečný počet kotev, nebo struktura odpovídající výpočtu presence detection (stav, kdy není k dispozici dostatek kotev k výpočtu, ale kdy nás zajímá pouze jestli byl na kotvě zachycen signál).

Výpočet výšky pomocí barometrů je v procesu výpočtu zahrnut na začátek před samotný výpočet horizontální polohy tagu. Zde jsou nejprve zkontrolovány nastavení serveru (tedy má-li být výpočet prováděn, či nikoliv), poté je spočítána výška relativně ke kotvám. Při úspěšném provedení výpočtu jsou do datové struktury tagu uloženy informace o této výšce a proces pokračuje výpočtem multilaterace. Celý tento výpočet se nachází také v části pro výběr ideálních kotev.

[8] `https://developer.gnome.org/glib/`
[9] `https://github.com/DaveGamble/cJSON`

Kapitola 5

Testování

Stejně jako předešlé kapitoly, může být testování rozděleno na testování vizualizační aplikace a testování barometrických výpočtů. Oproti předchozím kapitolám zde prohodím pořadí těchto dvou celků, jelikož barometrické výpočty poskytují data, která jsou ve výsledné aplikaci vizualizována.

5.1 Testovací zařízení

K testování byly vyhrazeny dva paralelní systémy běžící nezávisle na sobě. První systém byl serverový počítač DELL, podobný serverům, které bývají společností distribuovány. K tomuto serveru byly připojeny čtyři kotvy umístěné ve výšce 3 metrů. Druhý systém byl klasický notebook Lenovo se serverem spuštěným ve virtuálním prostředí Virtual Boxu. Tato konfigurace většinou slouží k testování a předvádění systému. K notebooku byla připojena 1 kotva umístěna ve výšce 0.929 metru (okenní parapet). Jelikož je to nedostačující počet k výpočtu multilaterace, byl server v režimu presence detection. Jak již bylo dříve zmíněno presence detection je režim, ve kterém nepotřebujeme počítat multilateraci, ale pouze detekujeme přítomnost tagu na základě přijetí signálu na některé kotvě.

Oba systémy sdílely 2 tagy, ležící při dlouhodobém testu ve výšce 0.929 metru (vedle presence detection kotvy systému Lenovo) nastavené následovně:

- obnovovací frekvence: 100 ms;
- verze hardware: 4.7;
- verze firmware: 3.118;
- profil: RF4, kanál 7;
- nastavení barometru: vysoká přesnost;
- akcelerometr: vypnutý (aby tagy při neaktivitě nepřešly do režimu spánku);

5.2 Výpočet vertikální polohy

Při testování hodnot vypočtených na základě barometrických údajů nás zajímá především správnost výpočtu, funkčnost řešení v různých situacích a především do jaké míry je řešení reálně použitelné.

5.2.1 Správnost výpočtu

První částí testování je ověření správnosti řešení. Toto ověření se dá provádět na reálných datech. Postačí zprovoznit řešení, tak jak bylo popsáno v předešlé kapitole. Po zprovoznění řešení je potřeba naměřit výsledky nějakou jinou formou než skrze testovanou aplikaci. K měření byl využit standardní laserový dálkoměr. Měření probíhalo na dříve zmíněném serveru DELL.

V testované buňce byly definovány všechny parametry pro úspěšnou vertikální lokalizaci (výška plánu, velikost lokalizační buňky, výšky jednotlivých kotev, výška tagu při kalibraci). Po nastavení parametrů se tag začne pohybovat vertikálním směrem v reakci na pohyb reálný. Měření byla prováděna na různých místech buňky v různých výškách. Tabulka 5.1 shrnuje jednotlivá měření. Tabulka 5.2 vykresluje chyby jednotlivých měření. Rozdíl mezi jednotlivými testy je rozdílná doba kalibrace tagu. Pro test A byl tag nastaven na vysílací periodu 100 ms, kalibrace tedy proběhla za jednu sekundu. Perioda u testu B byla nastavena na 1000 ms a u testu C na 1500 ms. Kalibrace u těchto dvou testů byla znatelně delší (odpovídá deseti odeslaným zprávám).

Pozice	Realita	Test A	Test B	Test C
střed buňky, nízko	0,00 m	-0,45 m	-0,07 m	-0,09 m
střed buňky, uprostřed	1,37 m	1,09 m	1,36 m	1,33 m
střed buňky, vysoko	2,20 m	2,11 m	2,16 m	2,24 m
okraj buňky, nízko	0,00 m	-0,36 m	-0,09 m	-0,11 m
okraj buňky, uprostřed	1,37 m	1,11 m	1,37 m	1,31 m
okraj buňky, vysoko	2,20 m	2,02 m	2,32 m	2,25 m
blízko kotvy, nízko	0,00 m	-0,19 m	0,08 m	-0,10 m
blízko kotvy, uprostřed	1,37 m	1,22 m	1,40 m	1,38 m
blízko kotvy, vysoko	2,20 m	2,05 m	2,27 m	2,24 m

Tabulka 5.1: Porovnání hodnot vypočtených a reálných po kalibraci tagu.

Pozice	Chyba A	Chyba B	Chyba C
střed buňky, nízko	0,45 m	0,07 m	0,09 m
střed buňky, uprostřed	0,28 m	0,01 m	0,04 m
střed buňky, vysoko	0,09 m	0,04 m	0,04 m
okraj buňky, nízko	0,36 m	0,09 m	0,11 m
okraj buňky, uprostřed	0,26 m	0,00 m	0,06 m
okraj buňky, vysoko	0,18 m	0,12 m	0,05 m
blízko kotvy, nízko	0,19 m	0,08 m	0,10 m
blízko kotvy, uprostřed	0,15 m	0,03 m	0,01 m
blízko kotvy, vysoko	0,15 m	0,07 m	0,04 m

Tabulka 5.2: Spočtené chyby pro jednotlivé testy.

Z výsledků vyplývá, že po delší kalibraci zařízení funguje řešení správně, tag reaguje na změny výšky a odchylka od skutečné vertikální polohy je menší než chyba pří výpočtu multilaterace. Ukázalo se jako vhodné u tagu nejprve nastavit delší periodu, zkalibrovat jej, a poté nastavit periodu odesílání jakou sami potřebujeme. Doba kalibrace má výrazný vliv na správnost výpočtu.

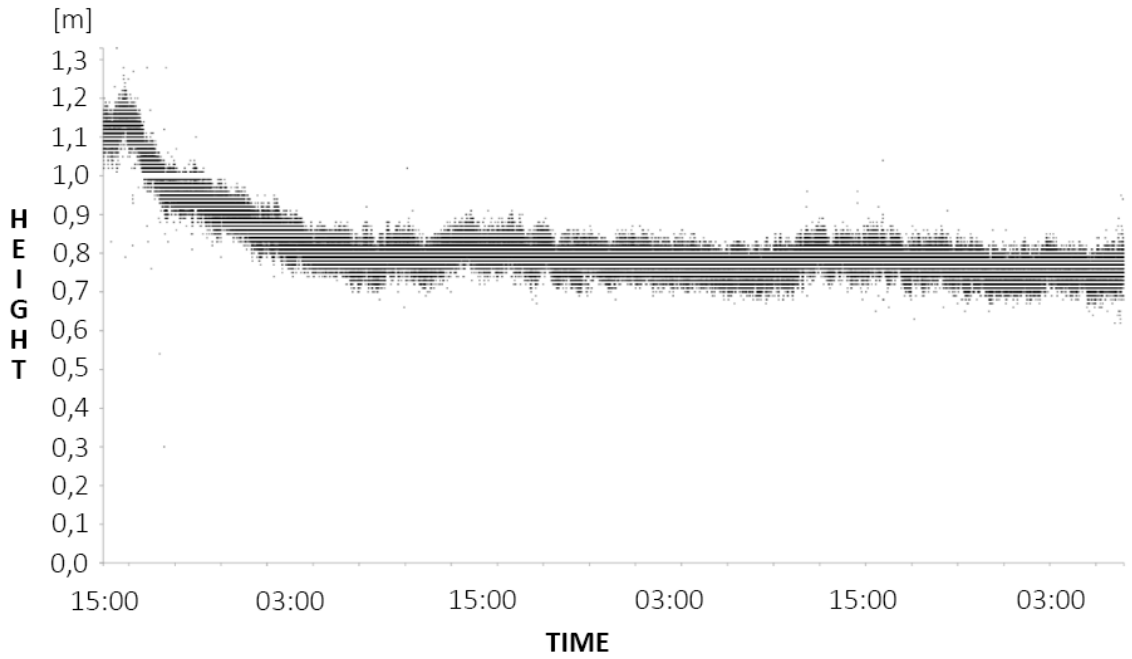
Dalším testem pro ověření správnosti výpočtu je kontrola funkčnosti přechodu mezi buňkami umístěnými nad sebou. Pro tento test byla zařízení přenesena na schodiště, kde mohou být buňky umístěny přímo nad sebou, dá se mezi nimi snadno přecházet a signál mezi buňkami není zcela přerušen.

5.2.2 Dlouhodobé testy

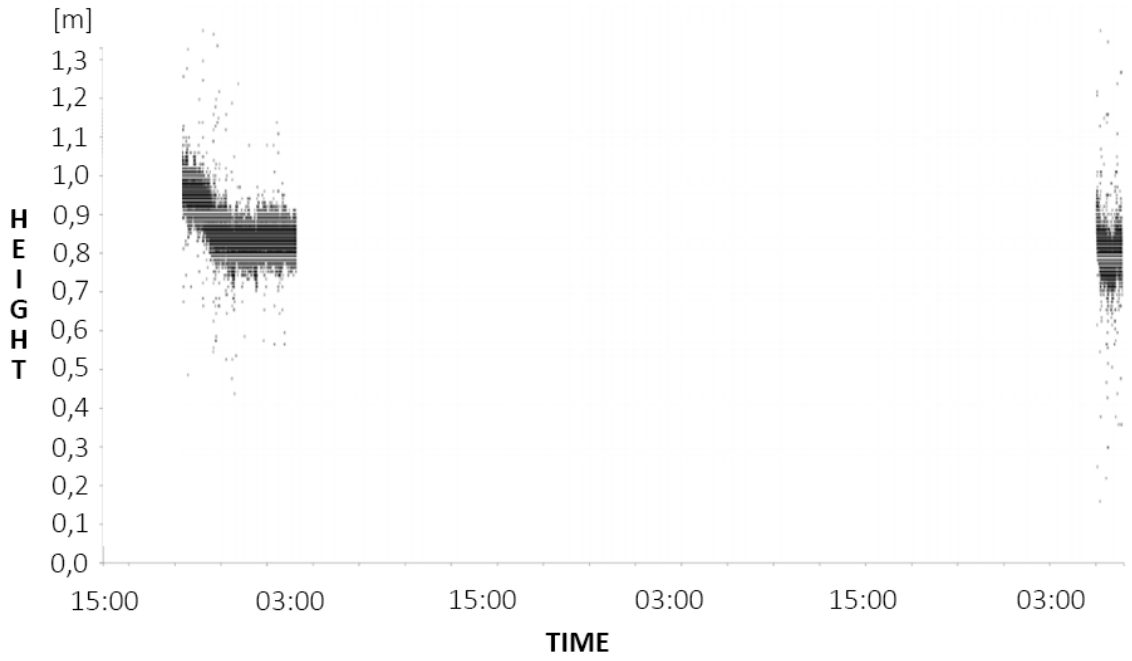
Žádoucím testem je také ověření toho, jak si barometrické senzory povedou v delším časovém úseku. Tímto testem bude zjištěno je-li vertikální lokalizace spolehlivá i po uplynutí určité časové doby od instalace systému a jednotlivých zařízení. Dále je třeba podotknout, že následujících výsledků je dosaženo u zařízení, která se celou dobu pohybují uvnitř lokalizovaného prostoru, a nejsou tedy vystavována změnám tlaku, kterým by nebyly vystaveny také referenční kotvy.

Tento test byl prováděn dvakrát, poprvé se jednalo o test několikadenní, kdy zařízení ležela v místnosti přes víkend, kdy nedocházelo k žádným prudkým změnám tlaku. Bohužel na zařízení Lenovo došlo k výpadku systému způsobenému nežádoucí aktualizací operačního systému. Část dat z tohoto důvodu chybí, nicméně výsledek má stejně průkazný charakter, jako by zařízení bylo v provozu celou dobu. Podruhé byl test spuštěn pouze na jednom zařízení, ale po dobu několika týdnů.

Výsledky prvního testu jsou vidět na obrázcích 5.1 a 5.2. Z testu vyplývá, že využití více kotev a vybírání referenční hodnoty mediánem má výrazný vliv na vyhlazení dat a snížení rozptylu (z 20 cm na 12 cm). Na virtuálním zařízení, kde byla použita pouze jedna referenční kotva, jsou data více rozptýlena. Dalším jevem, kterého si lze povšimnout, je, že nějakou dobu trvá, než se systém ustálí a poté se hodnoty vypočtené drží v okolí reálné hodnoty. Vodorovné pásy s chybějícími hodnotami jsou způsobeny zaokrouhlením hodnot a není třeba jim věnovat větší pozornost.

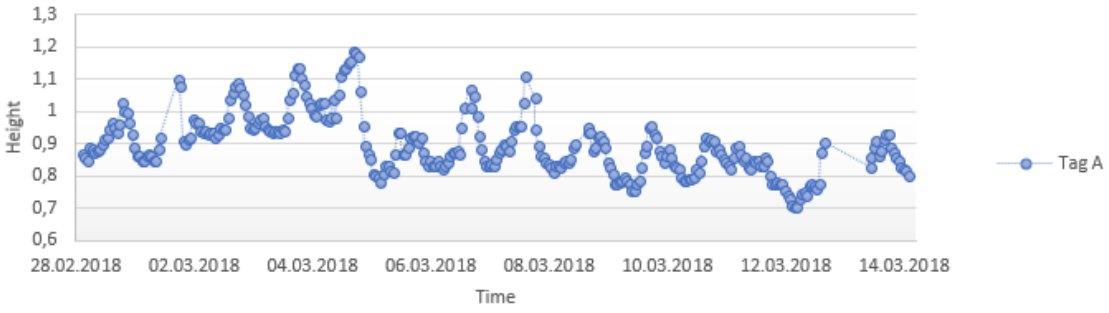


Obrázek 5.1: Krátký test na zařízení DELL (tag A).



Obrázek 5.2: Krátký test na zařízení Lenovo (tag A).

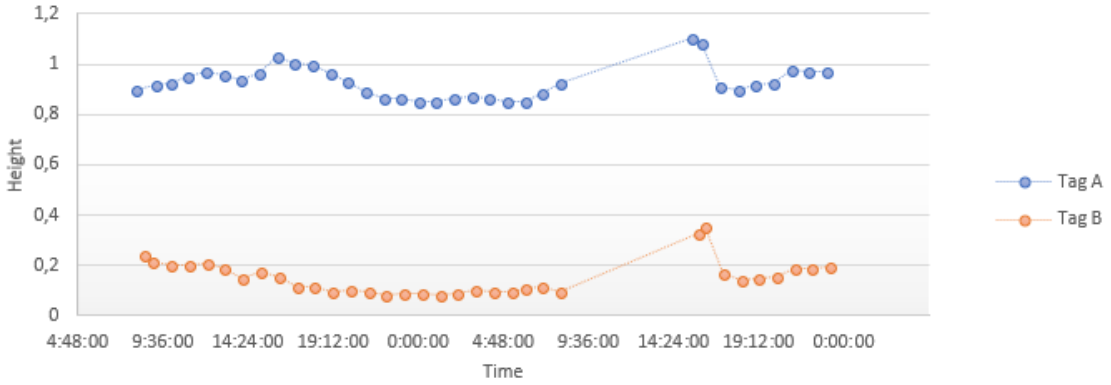
Několik úseků druhého testu je vidět na grafech 5.3, 5.4 a 5.5. Jelikož tento test trval mnohem delší dobu, jsou zaznamenaná data zprůměrována v hodinových intervalech. U druhého tagu na začátku měření došlo k výraznějšímu posunu výsledné hodnoty z důvodu vadné reakce barometrického senzoru na zahřívání. Oba tagy nicméně kopírují stejné výkyvy hodnot v důsledku změn tlaku v místnosti. Z grafů také vyplývá, že v hodinách, kdy se v místnosti nikdo nepohyboval (nebyla otevírána okna ani dveře), jsou hodnoty relativně ustálené. Chybějící data nejsou způsobena chybou serveru, ale krátkodobým využitím kotev k jiným účelům.



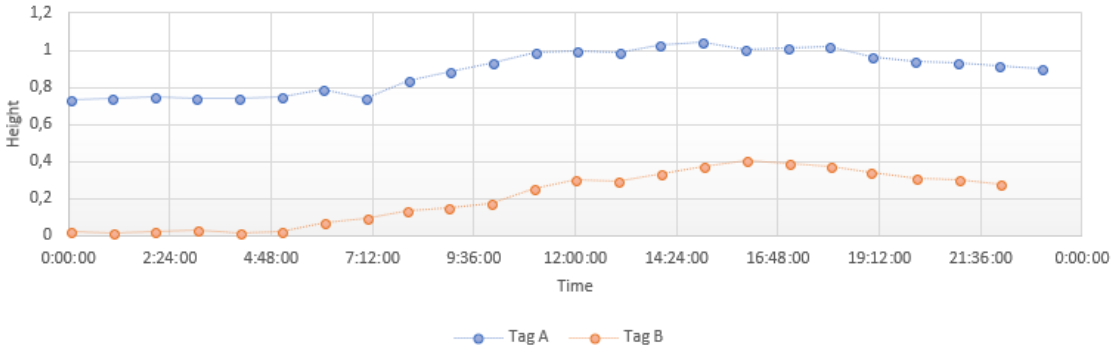
Obrázek 5.3: Dlouhý test na zařízení DELL.

5.3 Vizualizační aplikace

U vizualizační aplikace nás z pohledu testování zajímá především funkčnost řešení, pravdivost vizualizovaných dat a také, jak už to u 3D aplikací bývá, její výkon. Výkon hodnotíme



Obrázek 5.4: Úsek dlouhého testu s porovnáním obou tagů v průběhu dvou dnů.

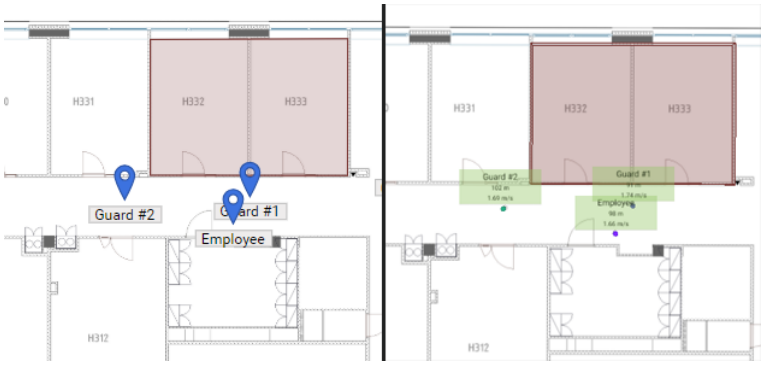


Obrázek 5.5: Úsek dlouhého testu s porovnáním obou tagů v průběhu jednoho dne na konci testování.

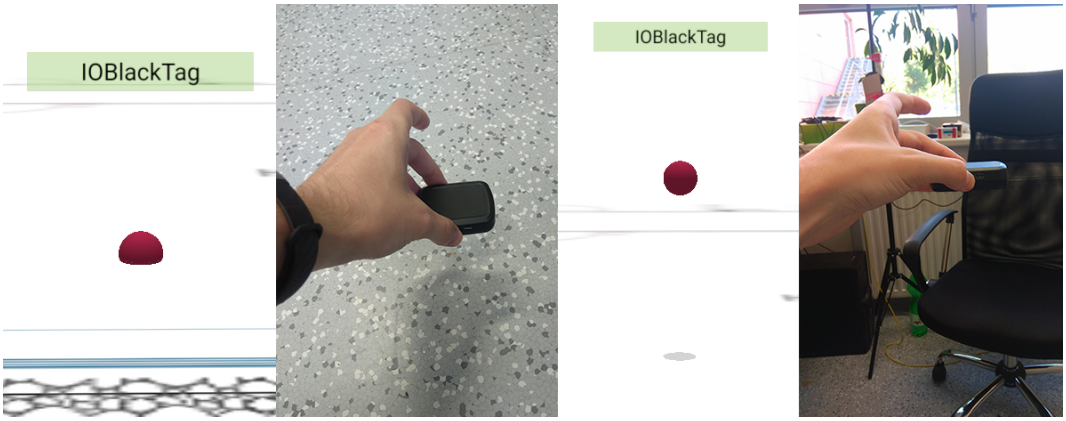
v rámci počtu vykreslovaných snímků, při určitém počtu vizualizovaných objektů. Aplikaci je vhodné vyzkoušet na různých zařízeních a v různých prohlížečích, i když byla primárně navržena pro internetový prohlížeč Google Chrome. V poslední řadě můžeme výsledky porovnat s původní vizualizační 2D aplikací, která trpěla právě nízkým výkonem při větším množství aktivních prvků systému.

5.3.1 Funkčnost řešení a pravdivost dat

To, do jaké míry jsou data vizualizována aplikací správná, tedy jsou-li všechny zařízení na správných místech plánu, popřípadě na správných plánech, odpovídá-li měřítko skutečnosti a jsou-li správně nastaveny odsazení počítaných poloh, můžeme ověřit porovnáním s aplikací původní, u které můžeme počítat se správností zobrazovaných dat, jelikož tato aplikace byla mnohokrát testována a nasazena v reálném provozu. Většina těchto testů byla provedena přímo při vývoji aplikace. Jednalo se jak o kontroly jednotlivých spočtených hodnot, tak o kontrolu vizuální, tedy jestli výstup aplikace odpovídá výstupu aplikace původní. Zde je třeba využít jisté míry představivosti, porovnáváme totiž aplikaci dvourozměrnou s její třírozměrnou variantou. Pro vizuální porovnání je k textu přiložen obrázek 5.6. Korelace mezi reálnou výškou a výškou ve virtuálním prostředí ilustruje obrázek 5.7.



Obrázek 5.6: Vizuální porovnání poloh z obou aplikací (vlevo původní, vpravo nová).



Obrázek 5.7: Porovnání reálné výšky s výškou virtuální.

5.3.2 Výkon aplikace

S obrovským počtem existujících zařízení, ať už se jedná o mobilní telefony, notebooky, nebo stolní počítače, přichází také jisté komplikace, co se kompatibility a funkčnosti řešení mezi těmito zařízeními týče.

Tabulka 5.3 porovnává výkon aplikace napříč různými prohlížeči v rámci jednoho zařízení. Zařízení, na kterém byly prováděny testy, je stolní počítač s procesorem Intel i5-3550 3,7 GHz, grafickou kartou Nvidia GeForce GTX 760 2GB a operačním systémem Windows 10. Počet snímků za vteřinu byl spočten z rozdílu časů naměřených na začátku a konci výpočetní smyčky aplikace. Tato hodnota byla zprůměrována z několika po sobě jdoucích hodnot. Měření probíhalo během pohybu kamery, což je nejnáročnější operace co se překreslování scény týče. Počet objektů byl ovlivněn množstvím zařízení a nastavením počtu vykreslovaných historických poloh.

Z výsledků je patrné, že nejlépe si s aplikací poradí prohlížeč Google Chrome, na který je aplikace cílená. Částečně je to způsobeno tím, že také knihovna Three.js je optimalizovaná právě pro tento webový prohlížeč. Maximální hodnota nepřesahuje 60 snímků za vteřinu, jelikož je tato hodnota nejběžnější obnovovací frekvence na většině standardních monitorů a je zbytečné tuto hodnotu překračovat, jelikož by výsledek nebyl pro uživatele patrný.

Výkon mezi různými zařízeními porovnává tabulka 5.4. Mimo jiné tabulka také zahrnuje srovnání výkonu původní vizualizační aplikace s aplikací implementovanou. Na všech zařízeních byla aplikace spuštěna ve webovém prohlížeči Google Chrome.

Prohlížeč	Počet objektů	Počet snímků za vteřinu
Google Chrome (verze 66.0)	5	60
Google Chrome (verze 66.0)	50	60
Google Chrome (verze 66.0)	500	52
Mozilla Firefox (verze 60.0.1)	5	60
Mozilla Firefox (verze 60.0.1)	50	58
Mozilla Firefox (verze 60.0.1)	500	33
Microsoft Edge (verze 42.17134.1)	5	60
Microsoft Edge (verze 42.17134.1)	50	60
Microsoft Edge (verze 42.17134.1)	500	46

Tabulka 5.3: Srovnání výkonu aplikace v různých prohlížečích s různým počtem objektů

Zařízení	Počet objektů	Původní aplikace	Nová aplikace
Stolní počítač	5	60 FPS	60 FPS
Stolní počítač	50	35 FPS	60 FPS
Stolní počítač	500	18 FPS	48 FPS
Notebook	5	52 FPS	60 FPS
Notebook	50	33 FPS	60 FPS
Notebook	500	16 FPS	42 FPS
Mobilní telefon	5	50 FPS	60 FPS
Mobilní telefon	50	22 FPS	58 FPS
Mobilní telefon	500	3 FPS	26 FPS

Tabulka 5.4: Srovnání výkonu původní aplikace s novou aplikací na různých zařízeních. Stolní počítač (grafická karta GeForce GTX 760 2 GB, 8 GB operační paměti, procesor Intel i5-3550 3,7 GHz) měl operační systém Windows. Notebook (grafická karta integrovaná, 8 GB operační paměti, procesor Intel i5-5200U 2,7 GHz) měl operační systém Ubuntu. Mobilní telefon (grafický procesor Adreno 506, 3 GB operační paměti, procesor Qualcomm Snapdragon 625) měl systém Android.

Naměřené výsledky ukazují pokles výkonosti u větších počtů objektů při využití původní aplikace. S novou aplikací nejsou tyto poklesy tak dramatické ani na méně výkonných zařízeních.

Kapitola 6

Závěr

V rámci první části diplomové práce bylo nastudováno API společnosti Sewio. Pro lepší srozumitelnost dat byly navrženy a aplikovány drobné úpravy a rozšíření tohoto API, například pro možnost ukládání 3D modelů. Dále byla podrobena analýze problematika senzorů a možností, kterými by mohly vylepšit současný stav lokalizace. V poslední řadě byl vytvořen návrh webové aplikace vizualizující interiérovou lokalizaci ve 3D.

Ve druhé části byla implementována 3D vizualizační aplikace s využitím knihovny Three.js. Hlavním cílem bylo vytvořit aplikaci, která si poradí s větším počtem zařízení beze ztráty plynulosti na co největším počtu zařízení. Z testování vyplývá, že se podařilo vytvořit aplikaci, která je skutečně lépe optimalizovaná než její předchůdce, a navíc zobrazuje scénu ve třech rozměrech. Funkčnost této aplikace byla ověřena porovnáním s aplikací původní. Mimo jiné je aplikace postavena nad architekturou MVC, tím se stává oproti svému předchůdci mnohem lépe rozšiřitelná.

Využitím senzorů, které se na lokalizovaných zařízeních nacházejí, byla vizualizace rozšířena o nové informace, které dále doplňují výsledný efekt aplikace. Pomocí akcelerometrů, gyroskopu a magnetometru byla vizualizace rozšířena o rotace jednotlivých zařízení. Tyto rotace jsou viditelné při importu uživatelských 3D modelů, což je jedna z dalších funkcionalit nového řešení. Tyto modely mohou být upravovány přímo v aplikaci skrze jednoduchý editor modelů.

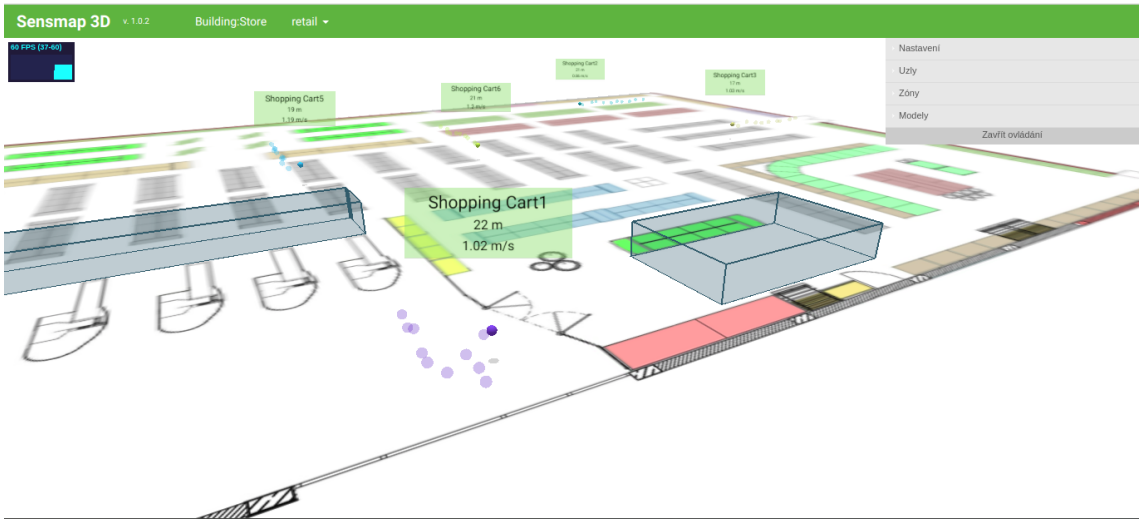
Poslední částí diplomové práce byla implementace výpočtu vertikální polohy lokalizovaných zařízení s pomocí barometrů nacházejících se uvnitř. Tento problém by se dal rozložit na dvě části, tedy na pochopení toho, jak funguje serverová část softwarového balíčku společnosti Sewio a samotnou implementaci těchto výpočtů společně se synchronizací jednotlivých zařízení a připravení dat, aby tento výpočet fungoval správně.

Po vytvoření funkčního řešení problému byla spolehlivost barometrických senzorů ověřena několika dlouhotrvajícími testy. Z testů vyplývá, že pro funkční senzor se chyba měření pohybuje v rozmezí několika desítek centimetrů, což je pro lokalizaci dostačující a vyrovná se chybě vyskytující se při výpočtu multilaterace. Ne všechny senzory však lze považovat za funkční. U některých senzorů došlo k posunu hodnot o více než 50 centimetrů, přičemž tyto hodnoty začínají být poměrně zavádějící. Testy byly prováděny na zařízeních, která se celou dobu vyskytují uvnitř stejné lokalizované buňky, takže nejsou připravena k tomu, aby opouštěla tento prostor a vracela se do něj. To může být pro mnoho instalací dostačující. V opačných případech je potřeba algoritmy dále vylepšovat a provádět další testy funkčnosti, což je jeden ze směrů, kterými je možné práci dále rozvíjet.

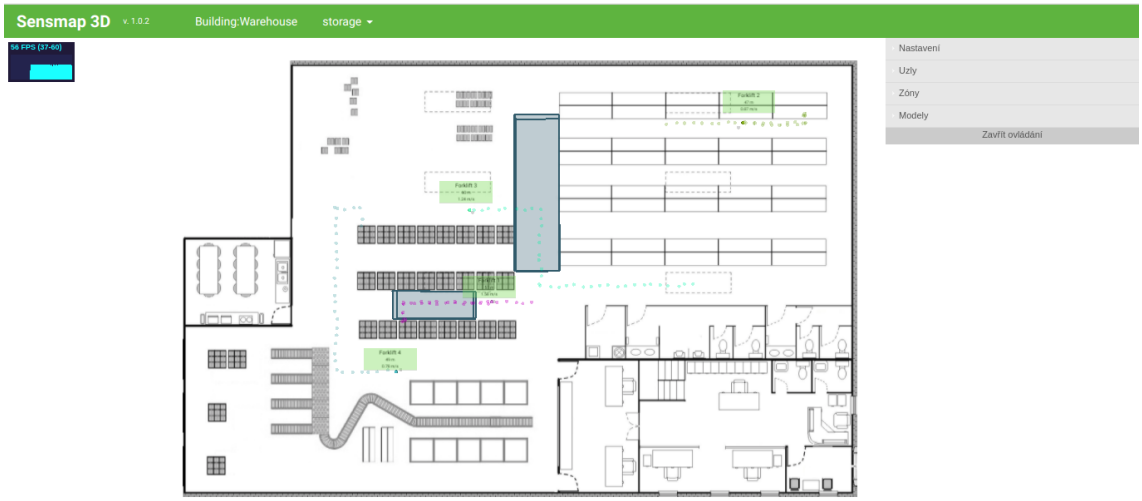
Co se budoucnosti implementovaného programu týče, nabízí se dvě možnosti, a to orientace čistě na zobrazování dat a prezentace. Zde by bylo třeba dále rozvíjet uživatelské

prostředí a nabídnout uživateli více možností, co se vizualizace týče (tepelné mapy historie pohybu, porovnávání statistik a jiné). Druhým směrem, kterým by se aplikace dále mohla ubírat, je jistý způsob spojení původní vizualizace s vizualizací novou. Toto spojení je žádoucí, jelikož původní aplikace stále slouží k nastavování velké části lokalizačního systému. Tato vizualizace by mohla být optimalizována pro větší instalace (například využitím 2D enginu nad WebGL) a dále spojena s vizualizací trojrozměrnou. Vznikla by tak jednotná univerzální aplikace.

Obrázky 6.1 a 6.2 zobrazují výslednou vizuální aplikaci v prostředí demonstračního portálu dostupného na adrese `http://rtlsstudio.com/studio/`. Aplikace na demonstračním portálu má nicméně svá omezení, protože je spuštěna na jednom serveru pro všechny příchozí uživatele. Hlavním omezením je nemožnost do mé aplikace vkládat 3D modely.



Obrázek 6.1: Výsledná aplikace při pohledu ze strany.



Obrázek 6.2: Výsledná aplikace při pohledu shora.

Literatura

[1] Bednář, J.: *Meteorologie: Úvod do studia dějů v zemské atmosféře*. Portál, 2003.

[2] Dempski, K.; Viale, E.: *Advanced lighting and materials with shaders*. Wordware Publishing, Inc., 2005.

[3] Diebel, J.: Representing attitude: Euler angles, unit quaternions, and rotation vectors. *Matrix*, ročník 58, č. 15-16, 2006: s. 1–35.

[4] Dirksen, J.: *Three. js essentials*. Packt Publishing Ltd, 2014.

[5] Gustafsson, F.; Gunnarsson, F.: Positioning using time-difference of arrival measurements. In *Acoustics, Speech, and Signal Processing, 2003. Proceedings. (ICASSP '03). 2003 IEEE International Conference on*, ročník 6, April 2003, ISSN 1520-6149, s. VI–553–6 vol.6, doi:10.1109/ICASSP.2003.1201741.

[6] Kremer, V. E.: Quaternions and SLERP. *Embots. dfki. de/doc/seminar_ca/Kremer_Quaternions. pdf*, 2008.

[7] Kuipers, J. B.; aj.: *Quaternions and rotation sequences*, ročník 66. Princeton university press Princeton, 1999.

[8] Lawrance, A.; Lewis, P.: An exponential moving-average sequence and point process (EMA1). *Journal of Applied Probability*, ročník 14, č. 1, 1977: s. 98–113.

[9] Lee, J. S.; Su, Y. W.; Shen, C. C.: A Comparative Study of Wireless Protocols: Bluetooth, UWB, ZigBee, and Wi-Fi. In *IECON 2007 - 33rd Annual Conference of the IEEE Industrial Electronics Society*, Nov 2007, ISSN 1553-572X, s. 46–51, doi:10.1109/IECON.2007.4460126.

[10] Leff, A.; Rayfield, J. T.: Web-application development using the model/view/controller design pattern. In *Enterprise Distributed Object Computing Conference, 2001. EDOC'01. Proceedings. Fifth IEEE International*, IEEE, 2001, s. 118–127.

[11] Li, B.; Harvey, B.; Gallagher, T.: Using barometers to determine the height for indoor positioning. In *Indoor Positioning and Indoor Navigation (IPIN), 2013 International Conference on*, IEEE, 2013, s. 1–7.

[12] Majerowicz, L.: MVC Pattern For Building Three.js Applications. [Online; navštíveno 01.01.2018].
URL `http://hecodes.com/2016/07/using-mvc-pattern-for-building-complex-three-js-applications/`

[13] Masse, M.: *REST API Design Rulebook: Designing Consistent RESTful Web Service Interfaces*. "O'Reilly Media, Inc.", 2011.

[14] McHenry, K.; Bajcsy, P.: An overview of 3d data content, file formats and viewers. *National Center for Supercomputing Applications*, ročník 1205, 2008: str. 22.

[15] Merino, B.: *Instant Traffic Analysis with Tshark How-to*. Packt Publishing Ltd, 2013.

[16] Parisi, T.: *WebGL: up and running*. "O'Reilly Media, Inc.", 2012.

[17] Prusty, N.: *Learning ECMAScript 6*. Packt Publishing Ltd, 2015.

[18] Shimrat, M.: Algorithm 112: position of point relative to polygon. *Communications of the ACM*, ročník 5, č. 8, 1962: str. 434.

[19] Watt, A. H.; Watt, A.: *3D computer graphics*, ročník 2. Addison-Wesley Reading, 2000.

[20] Whitehead, N.; Fit-Florea, A.: Precision & performance: Floating point and IEEE 754 compliance for NVIDIA GPUs. *rn (A+ B)*, ročník 21, č. 1, 2011: s. 18749–19424.

[21] Wolff, D.: *OpenGL 4.0 shading language cookbook*. Packt Publishing Ltd, 2011.

Příloha A

Obsah média

- /latex/ – složka obsahující zdrojové soubory technické zprávy;
- /src3d/ – složka obsahující zdrojové soubory vizualizační aplikace;
- /srcserver/ – složka obsahujicí zdrojové soubory serveru;
- /videos/ – složka obsahující videa demonstrující výslednou aplikaci formou krátkých návodů;
- /documentation.pdf – technická zpráva;
- /readme.txt – soubor obsahující návod na zprovoznění řešení;