VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV INFORMAČNÍCH SYSTÉMŮ
DEPARTMENT OF INFORMATION SYSTEMS

SIMULACE POHYBU MOBILNÍHO ZAŘÍZENÍ V RUCE ČLOVĚKA

MOVEMENT SIMULATION OF A HANDHELD DEVICE

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE **Bc. MAREK HÁK**
AUTHOR

VEDOUCÍ PRÁCE **Ing. RADEK HRANICKÝ, Ph.D.**
SUPERVISOR

BRNO 2024

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

Zadání diplomové práce

153709

Ústav:	Ústav informačních systémů (UIFS)
Student:	**Hák Marek, Bc.**
Program:	Informační technologie a umělá inteligence
Specializace:	Vývoj aplikací
Název:	**Simulace pohybu mobilního zařízení v ruce člověka**
Kategorie:	Bezpečnost
Akademický rok:	2023/24

Zadání:

1. Seznamte se s rozhraním Generic Sensor API pro přístup k senzorům v mobilních zařízeních pomocí webového prohlížeče. Seznamte se s rozšířením JShelter a způsoby, jakými chrání proti zneužití těchto dat.
2. Zkoumejte, jak se čtená data ze senzorů mobilního zařízení mění, když s ním člověk pohybuje. Porovnejte, jaká data jsou získávána, je-li zařízení v pohybu, v klidu, a pokud je použit nástroj JShelter v režimu simulace stacionárního zařízení.
3. Nastudujte způsoby, jakými je možné simulovat pohyb člověka.
4. Po konzultaci s vedoucím navrhněte řešení (algoritmus či model), které bude simulovat pohyb mobilního zařízení v ruce člověka.
5. Navržené řešení implementujte.
6. Integrujte vytvořené řešení do nástroje JShelter tak, aby simulovanou polohu využívaly moduly pro wrapping volání z Generic Sensor API.
7. Experimentálně ověřte použitelnost vytvořeného řešení a zhodnoťte dosažené výsledky.

Literatura:

- Lane, N.D. & Miluzzo, Emiliano & lu, Hong & Peebles, Daniel & Choudhury, Tanzeem & Campbell, A.T.. (2010). "A survey of mobile phone sensing." IEEE Communications Magazine, IEEE. 48. 140 - 150. 10.1109/MCOM.2010.5560598.
- MLAPandy, Marcus G. "Computer modeling and simulation of human movement." *Annual review of biomedical engineering* 3.1 (2001): 245-273.
- Xiang, Yujiang, Jasbir S. Arora, and Karim Abdel-Malek. "Physics-based modeling and simulation of human walking: a review of optimization-based and other approaches." *Structural and Multidisciplinary Optimization* 42.1 (2010): 1-23.
- Hranický Radek. "Protection against fingerprinting with Generic Sensor API". JShelter 2022. [online, cit.: 1. 9. 2023]: https://jshelter.org/sensorapi/

Při obhajobě semestrální části projektu je požadováno:
Body 1 až 4.

Podrobné závazné pokyny pro vypracování práce viz https://www.fit.vut.cz/study/theses/

Vedoucí práce:	**Hranický Radek, Ing., Ph.D.**
Vedoucí ústavu:	Kolář Dušan, doc. Dr. Ing.
Datum zadání:	1.11.2023
Termín pro odevzdání:	17.5.2024
Datum schválení:	30.10.2023

Abstrakt

Webové prohlížeče poskytují stránkám přístup k pohybovým senzorům na mobilních zařízeních, jako jsou telefony či tablety. Sdílená data mohou být zneužita ke sledování a identifikaci uživatelů. Rozšíření pro webové prohlížeče JShelter poskytuje ochranu proti takovému zneužití pohybových dat předáváním falešných hodnot. Tyto hodnoty však simulují statické zařízení, což může vést k detekci simulace. Cílem této práce je vytvoření simulace pohybu zařízení, která bude generovat uvěřitelný pohyb zařízení v rukou člověka. Před návrhem proběhla analýza dat ze senzorů a průzkum metod simulace pohybu. Pro generaci pohybu jsou využity sady parametrů generované genetickým algoritmem. Výsledné řešení bylo integrováno do rozšíření JShelter a experimenty ukázaly dobré výsledky a výpočetní nenáročnost řešení.

Abstract

Web browsers give websites access to motion sensors on mobile devices such as phones and tablets. The shared sensor data can be exploited to track and identify users. The JShelter browser extension provides protection against such exploitation of motion data by passing fake values. However, these values simulate a stationary device, which can lead to detection of the simulation. The goal of this thesis is to create a simulation that will generate believable device motion in the hands of a human. Prior to the design, sensor data analysis and exploration of motion simulation methods were conducted. Sets of parameters generated by a genetic algorithm are used for motion generation. The resulting solution was incorporated into the JShelter extension and experiments showed good results and performance of the solution.

Klíčová slova

Simulace pohybu, Pohybové senzory, Webové prohlížeče, Javascript, Ochrana soukromí, Webové rozšíření, Genetický algoritmus

Keywords

Movement simulation, Movement sensors, Web browsers, Javascript, Privacy protection, Web extension, Genetic algorithm

Citace

HÁK, Marek. *Simulace pohybu mobilního zařízení v ruce člověka.* Brno, 2024. Diplomová práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Ing. Radek Hranický, Ph.D.

Simulace pohybu mobilního zařízení v ruce člověka

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením pana Ing. Radka Hranického, Ph.D. Uvedl jsem všechny literární prameny, publikace a další zdroje, ze kterých jsem čerpal.

.....
Marek Hák
13. května 2024

Poděkování

Děkuji vedoucímu práce panu Ing. Radku Hranickému, Ph.D. za odborné vedení, ochotu a skvělý přístup při tvorbě této práce. Chtěl bych také poděkovat mé rodině a kamarádům za jejich podporu.

Obsah

Seznam obrázků

Kapitola 1

Úvod

Internetové prohlížeče předávají při komunikaci s webovými stránkami velké množství dat. Díky velkému množství těchto dat mohou provozovatelé stránek jednoznačně určit, o jakého člověka se jedná. Tento tzv. otisk (fingerprint) může být použit pro sledování zálib a zvyků daného člověka na dané stránce, ale i na jiných webových stránkach. Tato data mohou být použita pro vylepšení služeb poskytovaných uživateli, ale mohou být také využita např. pro cílenou reklamu nebo pro prodej zprostředkovatelům dat.

Prohlížeče využívající jádro Chromium (např. Google Chrome, Microsoft Edge nebo Opera) v základní konfiguraci dávají webovým stránkám přístup k senzorům zařízení pomocí sady objektů, kolektivně nazývaných Generic Sensor API. Nepříjemnou skutečností je absence jakéhokoliv upozornění, že stránka by chtěla používat či aktivně používá nějaký senzor. Existují však prohlížeče (např. Brave Browser), které se uživatele zeptají na povolení přístupu k senzorům pro danou webovou stránku.

Data ze senzorů je možné zneužít na vytvoření výše zmíněného otisku ze vzorů chování uživatele a nedokonalostí samotných senzorů. Také je možné pomocí sledování měnících se dat ze senzorů zjistit dodatečné informace o uživateli, které by jinak vyžadovaly explicitní zadání uživatelem. Takové informace mohou zahrnovat například pohlaví, přibližná lokace, vzor chůze člověka a další.

Z výše zmíněných důvodů je tedy zřejmé, že internetové stránky či komponenty třetích stran nacházející se na stránkách mohou pomocí pohybových senzorů značně narušit soukromí uživatelů. Pro ochranu soukromí je možné použít dva hlavní postupy. Prvním z nich je přístup k senzorům zakázat (globálně či pro jednotlivé stránky) nebo používat prohlížeč, který k nim přístup ani neumožňuje. Takovou ochranu lze však samotnou použít jako součást otisku uživatele. Druhou možností, kterou se zabývá tato diplomová práce, je stránkám předkládat falešné údaje generované tak, aby bylo obtížné je rozeznat od reálných dat. V ideálním případě bude soukromí uživatele chráněno, aniž by stránka tušila, že data jsou generována synteticky.

Cílem této práce je navržení simulace, která by generovala realistický pohyb zařízení v rukou člověka. Bude proveden průzkum metod pro generování pohybu, kterých by mohla výsledná simulace využívat. Data získávaná ze senzorů však nejsou perfektní, a tak bude provedena analýza dat senzorů, ve které budou nepřesnosti a charakteristiky těchto dat prozkoumány. Návrh simulace bude na těchto dvou částech stavět, ale v potaz je také nutné brát limitace a omezení výsledného řešení.

Navržené a implementované řešení bude stavět na simulaci kostry člověka. Kosti budou mít omezené maximální výchylky a pro generaci pohybu bude využito skládání sinusoid, které určují vychýlení každé z kostí. Pro generaci sad vhodných parametrů bude využito

genetického algoritmu. Výsledkem bude sada pohybů, která umožní použití různých pohybů pro různé webové stránky a také interpolaci mezi těmito pohyby v průběhu času. Pohyb kostry a vygenerované sady parametrů budou integrovány do webového rozšíření JShelter a proběhne experimentální průzkum a zhodnocení dosažených výsledků.

V kapitole 2 proběhne průzkum současného stavu senzorů na webu a ochrany soukromí dat senzorů. Kapitola 3 ukáže postupy a možnosti simulace lidského pohybu. Kapitola 4 popíše postup sběru a průzkumu dat. V kapitole 5 budou upřesněny požadavky a omezení pro výslednou simulaci a dále bude proveden její návrh. Kapitola 6 popisuje postup a detaily implementace navrženého řešení a v kapitole 7 proběhne ověření funkčnosti, analýza výkonu a zhodnocení dosažených výsledků.

Kapitola 2

Současný stav

Tato kapitola popisuje současný stav v oblasti senzorů mobilních zařízení, potenciálních rizik a možnosti ochrany vůči tvorbě otisků. Probraná témata budou použita v dalších kapitolách této práce. Nejprve se v části 2.1 popíše způsob přístupu k pohybovým senzorům zařízení na webu pomocí rozhraní Generic Sensor API. Dále budou v části 2.2 představena rizika senzorových dat a uvedeny doporučené strategie pro jejich zmírnění. V poslední části 2.3 bude prozkoumána možnost ochrany soukromí senzorových dat pomocí rozšíření prohlížeče JShelter.

2.1 Generic Sensor API

Generic Sensor API je množina JavaScript rozhraní poskytovaná webovými prohlížeči, umožňující sjednocený přístup k senzorům zařízení. Webové aplikace mohou data ze senzorů využívat pro pokročilé funkce jako počítání kroků, geolokaci či nové přístupy k interakci. Tato část je převzata z oficiální specifikace W3C [26].

V následujícím textu je pojmem *senzor zařízení* myšlen samotný senzor v zařízení, který snímá a poskytuje určitou fyzikální veličinu, zatímco *senzor* se vztahuje k třídám senzorů, které jsou poskytovány skrze Generic Sensor API.

2.1.1 Rozhraní senzoru

Základním definovaným rozhraním je *Sensor*. Toto rozhraní není používáno přímo, slouží jako základ pro přístup ke konkrétním senzorům. Rozhraní definuje:

- metody *start* a *stop*, které ovládají stav objektu,
- události (primárně *onreading*), které umožňují asynchronní interakci se senzorem,
- časové razítko (*timestamp*) posledního čtení daného fyzického senzoru v milisekundách,
- příznaky *activated* a *hasReading*, informující o stavu daného senzoru.

2.1.2 Přístup k senzoru

Pro přístup k některému ze senzorů je využíváno dané podtřídy rozhraní *Sensor*. Tyto podtřídy nemusí přímo vést na pouze jeden senzor zařízení. Poskytovaná data mohou být

získávána kombinací dat z několika různých senzorů zařízení (např. *RelativeOrientationSensor* využívá akcelerometr a gyroskop).

Nutností je zjištění podpory daného senzoru prohlížečem a jeho přítomnost v daném zařízení. Je také možné, že senzor přestane být dostupný (např. odepřením přístupu) v průběhu používání. Pro takovou situaci je nutné využít události *onerror*.

Při vytváření objektu senzoru je specifikována frekvence čtení. Tato frekvence však neovlivňuje rychlost čtení senzoru zařízení, pouze udává frekvenci události *onreading*. Díky tomu se přečtené hodnoty mezi vyvoláním dvou událostí *onreading* nemusí změnit.

2.1.3 Dostupné senzory

Generic Sensor API je v současnosti podporováno pouze v prohlížečích využívajících jádro *Chromium*. Senzory, které jsou v současnosti poskytovány a jejich příslušné senzory zařízení jsou popsány v tabulce 2.1. Od využívaných senzorů zařízení se také odvíjí potřebná povolení pro používaní daného senzoru. [17]

Tabulka 2.1: Dostupné senzory a jimi využívané senzory zařízení

Jméno senzoru	Využívané senzory zařízení
Accelerometer	akcelerometr
LinearAccelerationSensor	akcelerometr
GravitySensor	akcelerometr
Gyroscope	gyroskop
RelativeOrientationSensor	gyroskop a akcelerometr
AbsoluteOrientationSensor	akcelerometr, gyroskop a magnetometr
Magnetometer	magnetometr
AmbientLightSensor	snímač okolního světla

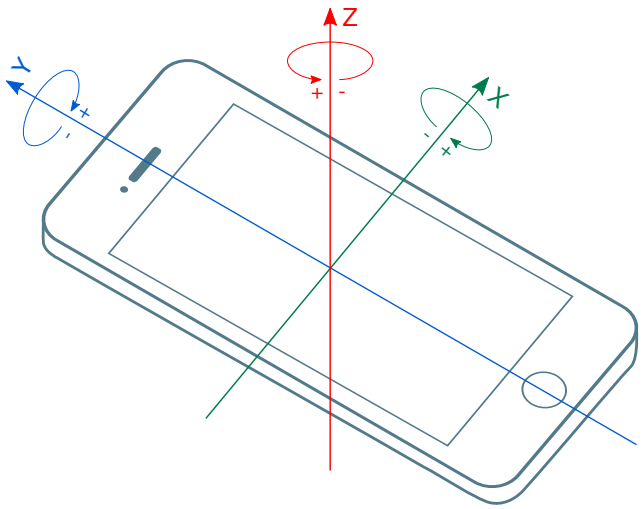
Ve způsobu použití se tyto senzory dělí na 3 skupiny:

1. snímač okolního světla obsahující pouze 1 hodnotu *illuminance*,
2. akcelerometr, senzor lineární akcelerace, senzor gravitace, gyroskop a magnetometr, které vracejí 3 hodnoty (x, y a z),
3. senzory orientace obsahující kvaternion popisující rotaci zařízení.

2.1.4 Souřadnicový systém senzorů

Hodnoty získané z pohybových senzorů jsou většinou relativní k současné orientaci zařízení (výjimkou je senzor absolutní orientace). Na obrázku 2.1 lze vidět náčrt zařízení s jednotlivými osami a jejich směry natáčení. Pro příklad bude popsána osa Z. Osa Z je kolmá k rovině obrazovky zařízení. Kladný pohyb směřuje na stranu zařízení, kde se obrazovka nachází (tj. na přední stranu zařízení), záporný pohyb směruje na stranu zařízení bez obrazovky (tj. na zadní stranu zařízení). Rotace ve směru hodinových ručiček při pohledu po kladném směru osy má kladné znaménko.

Senzory umožňují také nastavení referenčního rámce. Možné jsou dvě hodnoty – *device* (dle zařízení, výchozí hodnota) a *screen* (dle obrazovky). Při použití *device* je použit souřadnicový systém uvedený v předchozím odstavci. Pokud je použito referenčního rámce *screen*, mohou se zaměnit osy X a Y a jejich směry. V tomto režimu je souřadnicový systém měněn dle orientace obrazovky (např. při zobrazení obsahu na šířku).



Obrázek 2.1: Souřadnicový systém senzorů v režimu *device*, získáno z https://www.w3.org/TR/gyroscope

2.1.5 Použití senzoru

Ve výpisu 2.1 lze vidět jednoduché použití senzoru (akcelerometru). Výpis neobsahuje kontroly podpory, přítomnosti a povolení daného senzoru, podrobnější příklady užití lze nalézt v dokumentaci [17].

```
const senzor = new Accelerometer({ frequency: 30 });
senzor.addEventListener("reading", () => {
  console.log(`Akcelerace na ose X: ${senzor.x}`);
  console.log(`Akcelerace na ose Y: ${senzor.y}`);
  console.log(`Akcelerace na ose Z: ${senzor.z}`);
  console.log(`Časové razítko: ${senzor.timestamp}`);
});
senzor.start();
```

Výpis 2.1: Příklad použití akcelerometru v Generic Sensor API

2.2 Ochrana soukromí senzorových dat

Poskytnutí přístupu k senzorům zařízení může být vážné bezpečnostní riziko. Při použití více typů senzorů najednou či při dlouhodobém snímání se riziko může zvýšit. Tato rizika a doporučené kroky pro vývojáře webových prohlížečů a webových aplikací na zmírnění těchto rizik popisuje kapitola 4 specifikace Generic Sensor API [26], která je shrnuta v následujících podkapitolách.

2.2.1 Potenciální rizika

V této podkapitole je popsáno 5 primárních typů rizik ze specifikace Generic Sensor API [26] vyvstávajících ze současně dostupných senzorů.

Sledování uživatelů a tvorba otisků

Data ze senzorů lze použít k tvorbě otisku daného zařízení (tzv. *fingerprint*). Nedokonalosti vzniklé při výrobě jsou unikátní pro každý senzor. Tyto nedokonalosti lze detekovat a pomocí nich identifikovat dané zařízení napříč kontexty, jak je ukázáno v článku [25]. Dále je možné při současném čtení dat ze senzorů přes více kontextů zjistit shodu těchto dat a identifikovat tak stejné zařízení. Pomocí otisků je možné sledovat aktivitu uživatele napříč stránkami. Otisky je možné použít pro zlepšení uživatelské přívětivosti, ale i pro škodlivé účely.

Sledování pohybu

Pomocí sledování dat akcelerometru je možné odhadnout polohu uživatele bez použití GPS, ukázáno v konceptu *ACComplice* [8]. Statistické modely použijí sledovaná data pro odhad dráhy zařízení. Dráha je poté porovnána algoritmy s mapou a poloha zařízení odhadnuta s přesností až 200 m.

Identifikace uživatele

Pohybové senzory umožňují identifikaci uživatelů pomocí sledování způsobu chůze či podobných vzorů chování [3].

Odposlouchávání

Data z gyroskopu lze použít pro odposlouchávání konverzací bez přístupu k mikrofonu [15]. Tato data obsahují pouze nízké frekvence zvuku (pod 200 Hz), ale pomocí zpracování signálu a strojového učení je možné zjistit informace o mluvícím a také rekonstruovat řeč.

Sledování stisknutých kláves

Pomocí pohybových senzorů [14] a senzoru okolního světla [24] je možné detekovat zadaná hesla (PIN kód, kreslení vzoru). Útok by mohl využít např. vloženého rámce na citlivé stránce (internetové bankovnictví).

2.2.2 Strategie ochrany

Specifikace Generic Sensor API [26] popisuje několik obecných strategií, které by měly být dodrženy prohlížeči pro snížení rizik spojených se senzory:

- Bezpečný kontext - přístup k senzorům poskytuje silné možnosti a je explicitně specifikací považován za prioritní cíl pro útočníky, tudíž jsou rozhraní dostupná pouze v rámci bezpečného kontextu (HTTPS),
- Povolení přístupu - pro využívání senzorů musí jejich kontext pracovat s povoleními a musí mu být pomocí těchto povolení přístup umožněn,

- Aktivní oblast - čtení dat ze senzorů je umožněno pouze aktivním dokumentům, které mají stejný původ jako dokument právě aktivního prvku (tzv. *focused area document*).

Dále jsou doporučeny další strategie, které by měly být aplikovány na každý senzor jednotlivě, dle uvážení tvůrců a konrétního senzoru:

- omezení maximální vzorkovací frekvence,
- omezení přesnosti senzoru,
- omezení počtu poskytnutých výčtů dat,
- informování uživatele o současně používaných senzorech,
- pro krajní případy zastavení senzoru.

2.3 JShelter

JShelter je rozšíření pro webový prohlížeč Mozilla Firefox a prohlížeče využívající jádro *Chromium* (např. Google Chrome, Microsoft Edge, Opera a Brave Browser). Cílem tohoto rozšíření je detekce a prevence tvorby otisků, úprava webových API pro omezení možných útoků a prevence útoků zaměřujících se na získávání informací o zařízení, prohlížeči, uživateli a jeho okolí. Tato sekce je založena na článku o rozšíření JShelter [21].

2.3.1 Detekce tvorby otisků

Rozšíření je schopné detekovat pokusy o aktivní tvorbu otisku. Pro tento účel sleduje využití API často používaných pro tvorbu otisků a heuristickým přístupem detekuje sběr otisku. Pokud je pokus o vytvoření otisku detekován, uživatel je informován a dostane možnost zablokovat další asynchronní HTTP požadavky, které by mohly vést k odeslání jeho otisku. Pro každou stránku je možné nahlédnout do shrnutí výstupu detektoru otisků.

2.3.2 Úprava webových API

Rozšíření upravuje chování webových API, která jsou často používána k útokům nebo tvorbě otisku. Pro úpravu jsou používány 3 hlavní strategie, které jsou voleny případ od případu:

- snížení přesnosti,
- navrácení falešných hodnot,
- skrytí informace.

Nabízeny jsou 4 uživatelem volitelné úrovně, sahající od žádné ochrany až po přísnou ochranu, která zahrnuje zakázání některých API a největší úpravu navrácených hodnot. Tato nejvyšší úroveň však zvyšuje riziko tvorby otisku, protože generované falešné hodnoty jsou ve všech kontextech stejné.

2.3.3 Senzory

Součástí úpravy webových API je také úprava funkce senzorů Generic Sensor API. Hodnoty všech senzorů jsou generovány falešné. Snímač okolního světla simuluje stacionární zařízení ve vnitřním osvětleném prostředí. Pohybové senzory simulují plně stacionární zařízení. Orientace je generována pseudonáhodně pro každou doménu – senzory na stejné doméně simulují stejnou orientaci.

Vážnou zranitelností senzorů, kterou JShelter řeší, jsou časová razítka hodnot. Bylo zjištěno, že časová razítka počítají počet milisekund od chvíle, kdy bylo zařízení zapnuto. Čas zapnutí zařízení se dá považovat za téměř unikátní hodnotu pro každé zařízení, a tudíž je velmi silnou informací pro tvorbu otisku. Rozšíření tento problém řeší nastavením počáteční hodnoty dle času vytvoření kontextu dané stránky.

2.3.4 Změny v JavaScript prostředí

JShelter upravuje JavaScript prostředí prohlížeče monitorováním a upravováním výsledků vestavěných API a chování vestavěných objektů. Toho je dosaženo obalením potřebných částí. Ke vložení musí dojít v každém kontextu prohlížeče, tj. při vytvoření nového okna, nové záložky a vložení rámce. Vytvořené řešení používá pouze společné API prohlížečů, není tedy vázané na jednotlivé prohlížeče. Metody jsou upraveny co nejvýše v prototypovém řetězci.

Další implementovanou ochranou je tzv. *Network Boundary Shield* (štít hranice sítě), který chrání proti útokům směrovaným z vnější sítě (internetu) na lokální síť. Ochrany je dosaženo pomocí monitorování a filtrování požadavků HTTP. Příkladem možného útoku je skenování otevřených portů a využití této informace pro tvorbu otisku.

Kapitola 3

Průzkum simulace pohybu

Simulace pohybu člověka je rozsáhlé a stále se vyvíjející téma. Hlavními obory, které se tímto tématem zabývají, jsou robotika, biomechanika a počítačová animace. Hlavní způsoby, které v této části budou představeny, jsou kinematické simulace, dynamické simulace a simulace založené na datech. Nejprve bude představen model kostry, který je používán pro účely simulace. Dále bude následovat průzkum předem zmíněných způsobů simulace. Vybraná simulace bude sloužit pro simulaci lidského pohybu, který by potenciální hrozby na soukromí uživatelů dostatečně oklamal a působil realisticky.

3.1 Kostra

Pro účely simulace pohybu velmi komplexního modelu, kterým je v tomto případě člověk, se typicky tento model zjednoduší na jeho kostru. Po provedení simulace této kostry se na vygenerovaná data aplikují další potřebné operace pro daný účel.

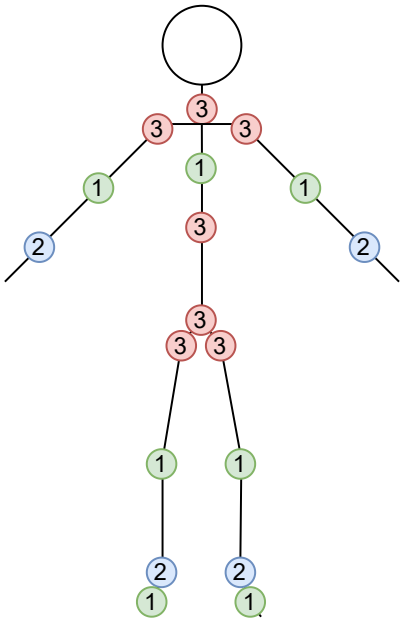
Kostra je složena z kostí pevné délky, které jsou spojeny klouby. Tyto klouby mohou mít různý počet stupňů volnosti. Pro kostru je možné specifikovat další vlastnosti a omezení, jako například maximální výchylky pro určité osy či pozice svalů, které mohou na kostru působit silami.

Na obrázku 3.1 lze vidět jednoduchou verzi takové kostry. Kosti (čáry) jsou spojeny klouby (kruhy) s určitým počtem stupňů volnosti. Toto je pouze příklad kostry, je možné využít koster s větší či menší složitostí a přesností (vzhledem k pravé kostře člověka).

V kostře jsou často definovány body, které pro nás v simulaci mají větší význam než ostatní. Takové body jsou v práci [2] nazvány *koncové efektory*. Koncové efektory mohou být např. chodidla, dlaně či hlava. Různé způsoby simulací aplikují na koncové efektory dodatečné požadavky.

3.2 Kinematické simulace

Kinematické simulace pohybu jsou založeny na znalostech o pohybu a omezení daného modelu. Kinematické simulace nezohledňují síly či jiné fyzikální prvky, které k pohybu vedou. Stav modelu je generován z geometrických vlastností jednotlivých prvků (pozice, rychlost, zrychlení) v průběhu času. [4]



Obrázek 3.1: Zjednodušený model kostry člověka se zobrazenými klouby a jejich stupni volnosti

3.2.1 Dopředná kinematika

Postupy dopředné kinematiky [18] specifikují stav modelu v průběhu času. Pomocí definovaných úhlů a úhlových rychlostí kostry je zjištěna poloha a rychlost výsledného bodu. Počet těchto stavů je většinou malý a dodatečné stavy jsou vytvářeny interpolací mezi těmito klíčovými stavy. Kvalita simulace závisí na zvolených interpolačních technikách a na kvalitě vytvořených klíčových stavů. Jediným způsobem, jak může dopředná kinematika selhat, je pokud některý z předaných parametrů odporuje stanoveným limitům daného modelu [2].

3.2.2 Inverzní kinematika

Postupy využívající inverzní kinematiku specifikují pozice a orientace cílových bodů (typicky koncových efektorů) a z těchto parametrů vypočítají parametry modelu, který vyhovuje požadavkům [18]. Na rozdíl od dopředné kinematiky může být stavů modelu, který tyto požadavky splní, nekonečně mnoho, jeden, či žádný v závislosti na požadovaných polohách a orientacích koncových efektorů. Získání řešení tedy není vždy jednoduchým postupem.

Inverzní kinematiku je možné použít společně s dopřednou kinematikou [18]. Na model se nejprve aplikuje dopředná kinematika, čímž jsou získány základní pozice a orientace koncových efektorů, pro které jsou definována omezení. Omezením může být např. požadavek na spojení chodidla se zemí bez průniku do země. Aplikací těchto omezení na základní koncové efektory získáme jejich požadované pozice a orientace. Posledním krokem je aplikace inverzní kinematiky pro získání finálního stavu modelu.

3.3 Dynamické simulace

Dynamické simulace jsou založeny na simulaci sil působených svaly, působení okolního prostředí a gravitace. Lze tedy mluvit o fyzikálním přístupu, který se snaží s velkou přesností

simulovat okolní i vnitřní prvky a z vygenerovaných dat popsat kinematický stav modelu [4]. Dynamická simulace umožňuje více realistickou simulaci pohybu, pokud je potřeba aplikovat dodatečné požadavky na způsob pohybu. Takové požadavky mohou zahrnovat např. simulaci nošeného břemene, komplexního terénu či vlivu větru.

Stejně jako u kinematických simulací existuje dopředná a inverzní dynamika. Metody dopředné dynamiky získávají pohyb ze sil aplikovaných na model a metody inverzní dynamiky vypočítávají síly, které by daný pohyb modelu způsobily [18]. Následuje představení některých využití a metod dynamických simulací pohybu, především pro lidskou chůzi.

3.3.1 Dynamická simulace jako omezení

Výhod metod dynamických simulací je možné využít i bez nutnosti specifikace či plné simulace pohybu těmito metodami [18]. Pohyb je možné nejprve vytvořit pomocí kinetických modelů a poté zkontrolovat jeho validitu dynamickými metodami. Taková omezení mohou zahrnovat např. rovnováhu, komfort či kontrolu námahy kloubů. Pokud je některé z těchto omezení porušeno, může být na model aplikována korekce.

3.3.2 Model inverzního kyvadla

Lidská chůze zahrnuje kyvadlovou výměnu mezi potenciální energií a kinetickou energií. Kvůli tomu je model inverzního kyvadla často používán pro simulaci chůze člověka [27]. Výhodou je jednoduchost a uzavřené analytické řešení pro trajektorii těžiště modelu. Nevýhodou je nedostatečný dynamický model, kvůli kterému je obtížné touto metodou generovat realistickou a uvěřitelnou lidskou chůzi.

Jednoduchý model inverzního kyvadla, poprvé použit pro tvorbu dvounohého robota ve výzkumu Kajita a Tani [10], má několik předpokladů, které limitují schopnost generace uvěřitelné chůze:

- Těžiště má konstantní výšku a pohybuje se pouze v jedné dimenzi.
- Tělo je vždy podepřeno právě jednou nohou.
- Veškerá hmotnost je soustředěna v těžišti, a model tedy neuvažuje hmotnost noh.

Byly navrženy modely inverzního kyvadla, které tyto předpoklady odstraňují a umožňují pokročilou generaci chůze. Model navržený ve výzkumu Kajita, Matsumoto a Saigo [9] uvolňuje omezení pohybu pouze v jedné dimenzi (umožňuje tedy pohyb těžiště po rovině) a jeho pohyb zahrnuje i dvojitou oporu. Dále byl v práci Kudoh a Komura [12] popsán model inverzního kyvadla, který bere v potaz úhlový moment kolem těžiště. Byly také vytvořeny modely [20, 1], které aplikují vliv dynamik dalších objektů, jako je např. druhá noha.

3.3.3 Pasivní dynamická chůze

Doposud představené simulace pracovaly s předpokladem, že jejich model je schopen ovládat úhly svých kloubů. Pasivní dynamická chůze pracuje s předpokladem, že model své klouby neovládá [27]. Modely simulované touto metodou jsou schopné chůze z kopce připomínající člověka.

Modely pasivní dynamické chůze jsou kostry chodící po dvou končetinách, které mohou být poháněny čistě gravitací při pohybu i po mírném svahu, s malou nebo žádnou schopností ovládání svých kloubů [27]. Nohy se přirozeně chovají jako kyvadla a zachování úhlového momentu řídí kontakt kývající se nohy s povrchem.

Obrázek 3.2: Značky využívané pro získávání pohybových dat. Fotografie z https://www.flickr.com/photos/thecleversheep/5700379379

3.3.4 Metody na základě ZMP

Bod nulového momentu (*Zero-moment-point*, *ZMP*) je takový bod na zemi, kde se celkové horizontální síly setrvačnosti působící na model rovnají nule [5]. Nachází-li se tento bod uvnitř plochy tvořené chodidly (v případě dvojité opory i oblastí mezi nimi), model je v rovnováze.

V dynamických simulačních metodách, které jsou na ZMP založeny, je trajektorie ZMP předem naplánovaná. Taktéž pozice nohou podél této trajektorie je předepsaná. Následně je vypočítána trajektorie pánve (v případě lidské kostry) takovým způsobem, aby splňovala podmínku stability při následování trajektorie ZMP. Posledním krokem je získání potřebných úhlů kloubů použitím metody inverzní kinematiky. [27]

3.4 Pohybová data

Předchozí přístupy pro simulaci dat vyžadují empirickou znalost možností a charakteristik daných modelů (člověka). Pro odstranění této nutnosti a potenciální zvýšení uvěřitelnosti výsledné simulace je možné využít snímaných pohybových dat.

Získávání pohybových dat je typicky prováděno připevněním sledovaných značek na tělo subjektu, příklad takových značek lze vidět na obrázku 3.2. Sada dvou či více kamer sleduje subjekt a snímá jeho pohyby. Dvourozměrná data o připevněných značkách získaná těmito kamerami jsou sloučena programem, jehož výstupem je trojrozměrná poloha značek v průběhu snímání. Pomocí těchto poloh je poté možné sestavit průběh stavu kostry, na který je poté možné aplikovat další dynamické či kinematické postupy pro zpřesnění simulace a získání výsledného pohybu [11].

V posledních letech byly zkoumány a vyvíjeny přístupy pro snímání pohybu bez použití značek a v jiném než laboratorním prostředí [19]. To přináší výhody pro případy, ve kterých by vědomí o prováděném experimentu mohlo mít na subjekt nežádoucí vliv.

Nasnímaná pohybová data je možné, jak již bylo zmíněno, použít pro kinematickou či dynamickou simulaci. Při větším počtu charakteristických dat je možné mezi pohyby prolínat a vytvářet tak nové pohyby [18]. Dále lze získaná data využít pro trénování modelů využívající strojové učení pro imitaci lidských pohybů [23].

Kapitola 4

Sběr a průzkum dat

Pro účely této práce bude zapotřebí analyzovat data získaná ze senzorů. Budou sledovány jejich změny, přesnost a vlastnosti, které by poté simulace měla napodobovat. Pro sběr dat byla v rámci této práce vytvořena webová stránka. Získaná data byla analyzována a pro dodatečný průzkum dat byla vyvinuta aplikace, která zobrazuje nasbíraná data v prostoru.

4.1 Použité technologie

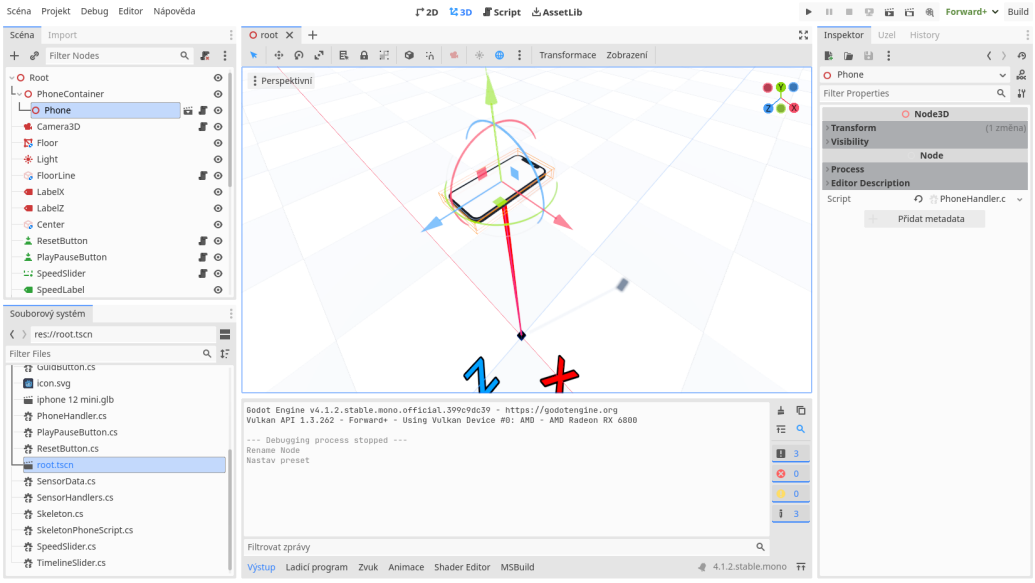
V této části budou představeny technologie, které byly použity pro tvorbu aplikace na sběr dat a aplikace pro zobrazení prostorových dat.

Blazor Pro vytvoření webové aplikace na sběr bylo využito frameworku Blazor, který umožňuje vývoj webových aplikací s použitím technologií .NET a jazyku C#. Umožňuje vykreslování na straně serveru i klienta, přičemž pro tuto aplikaci bylo využito vykreslování na klientovi, také známé jako *Blazor WebAssembly*, díky čemuž bylo možné využít statického hostování.

Progresivní webové aplikace Pro rozšíření možností aplikace na sběr dat byla vyvinuta jako progresivní webová aplikace (*Progressive Web App*, *PWA*). PWA využívají technologie webové platformy, které umožňují přiblížení se ke schopnostem aplikací vyvinutých přímo pro dané platformy [16]:

- Možnost instalace aplikace na zařízení,
- Ukládání obsahu aplikace do mezipaměti,
- Fungování bez připojení k internetu,
- Služby na pozadí,
- Zasílání upozornění uživatelům.

Vyvinutá aplikace využívá PWA k umožnění instalace aplikace do zařízení a fungování bez aktivního připojení k internetu.



Obrázek 4.1: Rozhraní prostředí Godot Engine

GitHub Pages GitHub Pages umožňuje hostování statických webových stránek a aplikací. Soubory webového obsahu, který bude hostován, jsou získány z GitHub repozitáře, přičemž mohou volitelně projít procesem sestavení. Při využívání GitHub Pages zdarma musí být repozitář, který je hostovaný, nastaven jako veřejný. Stránky mohou být hostovány na vlastních doménách nebo na doméně *github.io*. [6]

Supabase Pro uložení dat je využito platformy Supabase. Supabase umožňuje vytvořit server–klient aplikaci bez nutnosti tvorby vlastní serverové části. Pro každý projekt poskytuje databázi (*PostgreSQL*), autentifikaci, uložiště a další. Umožňuje připojení pomocí klientských knihoven (dostupné např. pro JavaScript, .NET, Flutter a další) nebo pomocí REST API. Nabízí několik možných úrovní služeb, které zahrnují i možnost využít služby bez placení.

Godot Engine Godot je všestranný open-source 2D a 3D herní engine určený pro různé druhy projektů. Je možné jej využít pro tvorbu her a aplikací určených pro stolní počítače, mobilní zařízení nebo web. Pro tvorbu aplikací je možné využít jazyk C# nebo *GDScript*, který je vytvořený přímo pro Godot a syntaxí připomíná jazyk Python. Pro tento projekt bylo využito jazyku C#. Vývojové prostředí obsahuje velké množství vestavěných nástrojů, které lze vidět na obrázku 4.1. [7]

4.2 Sběr dat

Sběr pohybových dat probíhal formou webové aplikace, která byla vytvořena ve frameworku Blazor a hostována na službě GitHub Pages. Získaná data jsou zasílána do databáze na platformě Supabase.

Frekvence použitých senzorů byla nastavena na 60 měření za sekundu. Data byla získávána ze senzorů akcelerace (akcelerometr, gyroskop, senzor gravitace a senzor lineárního zrychlení) a senzorů orientace (relativní a absolutní). Magnetometr není ve výchozím stavu

Tabulka 4.1: Přehled dat získaných z měření

Typ pohybu	Počet měření	Průměrná délka měření
Mobil na stole	11	41.07 s
Sezení	20	36.03 s
Chůze	27	29.90 s
Běh	1	18.48 s
Jízda autem	5	34.38 s
Chůze do schodů	2	17.54 s
Chůze ze schodů	1	14.24 s
Celkem	**66**	**33.00 s**

prohlížečů dostupný, jelikož je označen jako experimentální. Sběr probíhal v době mezi 28. 10. 2023 až 22. 12. 2023.

4.2.1 Sbíraná data

Sbíraná data se dělí na dvě kategorie – povinná a volitelná. Povinná kategorie zahrnuje samotná data sbíraná ze senzorů, čas započetí a ukončení měření a zvolený typ pohybu. Volitelná data může uživatel zadat před začátkem měření a zadané hodnoty jsou uloženy a automaticky vyplněny při příštím měření. Volitelná data zahrnují:

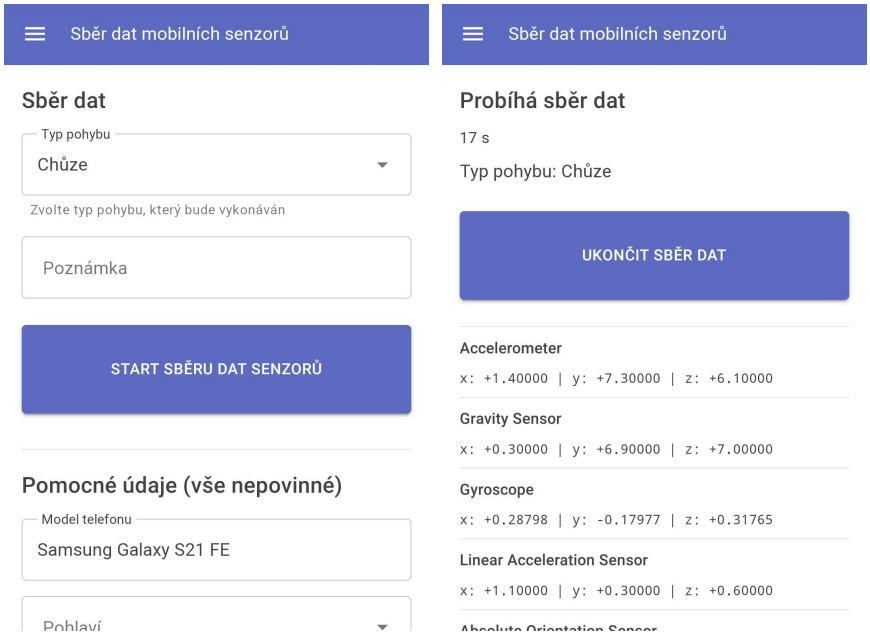
- model zařízení,
- pohlaví,
- věk,
- výšku,
- váhu,
- poznámku k měření.

4.2.2 Postup měření

Účastníci k webové aplikace přistoupí přes prohlížeč, který podporuje a poskytuje přístup k senzorům přes rozhraní Generic Sensor API. V současnosti to jsou pouze prohlížeče založené na jádře Chromium na mobilních zařízeních Android.

Úvodní stránka obeznámí účastníky s tématem této diplomové práce a se sbíranými daty. Následuje příprava měření, kterou lze vidět na levé straně obrázku 4.2, kde účastník vybere typ pohybu, který bude vykonávat. Dále může volitelně přidat poznámku k měření či volitelně sdělit upřesňující údaje, které by mohly být užitečné v rámci této práce (např. model telefonu nebo pohlaví).

Stránka samotného měření, kterou lze vidět na pravé straně obrázku 4.2, obsahuje ukazatel délky měření a současné hodnoty senzorů. Po ukončení má účastník na výběr, zda chce nasbíraná data odeslat ihned, či je uložit lokálně a zaslat ručně později. Pro tento účel je poslední dostupnou stránkou historie provedených měření, kde je možné nezaslané měření zpětně zaslat.



Obrázek 4.2: Rozhraní aplikace sběru dat

Přehled získaných data lze vidět v tabulce 4.1. Typů pohybu ke zvolení bylo více, než je v tabulce uvedeno, ale pro většinu z nich nebylo získáno žádné měření. Zaměřeno bylo na měření chůze a sezení. Počet získaných dat je nižší, ale pro účely této práce není vysoký počet měření nutný.

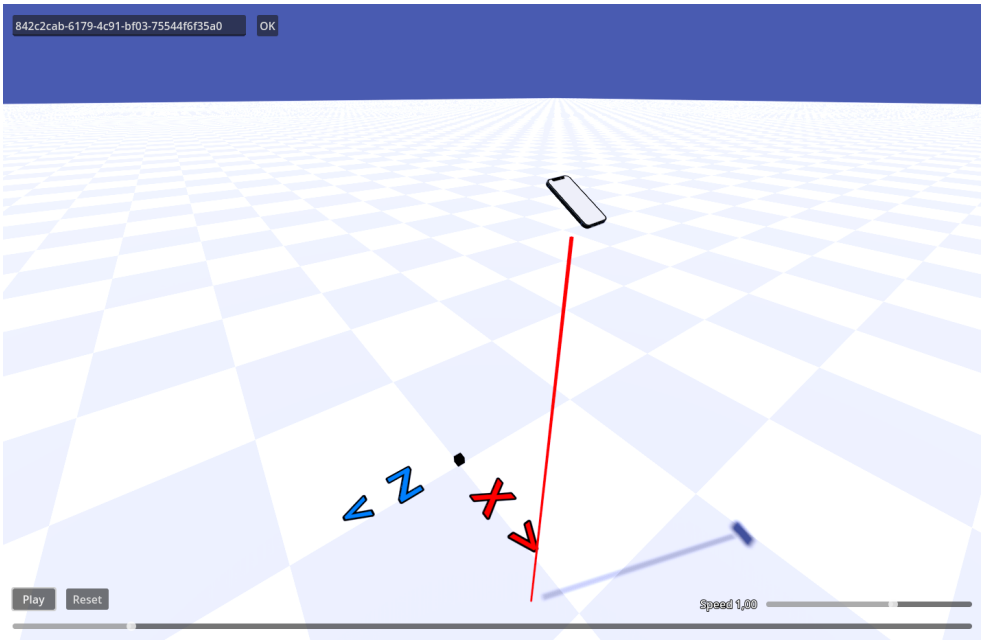
4.3 Prostorové zobrazení získaných dat

Data získaná ze senzorů mohou být velmi komplexní. Pro zjednodušení jejich průzkumu byl v rámci této práce vyvinut nástroj, který umožňuje jejich zobrazení v prostoru. Aplikace využívá herní engine *Godot*. Data jsou získávána přímo z databáze používané ke sběru dat. Po zadání identifikátoru vyžadovaného měření jsou načtena potřebná data. Ukázku aplikace lze vidět na obrázku 4.3.

Při načtení dat jsou předem vypočítány orientace a polohy zařízení pomocí dat ze senzorů relativní orientace a lineárního zrychlení. K vypočítaným hodnotám jsou přidány jejich časové značky. Záznam dat je poté možné přehrávat v libovolné rychlosti a směru či v časové linii záznamu přeskakovat a vracet se.

Získání orientace zařízení z nasbíraných dat je jednoduché. Senzor relativní orientace navrací přesnou lokální orientaci zařízení. Jediným problémem byl odlišný souřadnicový systém používaný v Generic Sensor API a v Godot.

Získání polohy je oproti orientaci obtížné. Protože není dostupný senzor, který by obsahoval rychlost zařízení, je nutné k získání údajů o pohybu použít senzor lineárního zrychlení. Pomocí zrychlení je však velmi obtížné zjistit přesnou rychlost zařízení. Hlavními důvody jsou neznámá počáteční rychlost zařízení a kumulativní chyba v důsledku velmi nízkého rozlišení senzorů – desetiny m/s^2. Pokusy o použití zrychlení pro získání rychlosti a polohy zařízení dopadly neúspěšně. Zařízení se postupem času typicky začalo pohybovat určitým směrem, a nakonec i opustilo prostor simulace. Způsobem jak tento problém zmírnit bylo průběhem času na rychlost zařízení aplikovat tlumení. Zobrazená poloha v aplikaci tedy ne-



Obrázek 4.3: Ukázka aplikace pro zobrazení dat senzorů v prostoru

ukazuje skutečnou polohu zařízení, ale slouží pouze k vizualizaci zrychlování a zpomalování zařízení.

4.4 Datová analýza

Získaná data byla prozkoumána pomocí tvorby a analýzy grafů a popisných statistik a využitím nástroje pro zobrazení dat, popsaného v sekci 4.3. Zobrazená data[1] jsou částí dat chůze získaných z aplikace pro sběr dat, popsané v sekci 4.2.

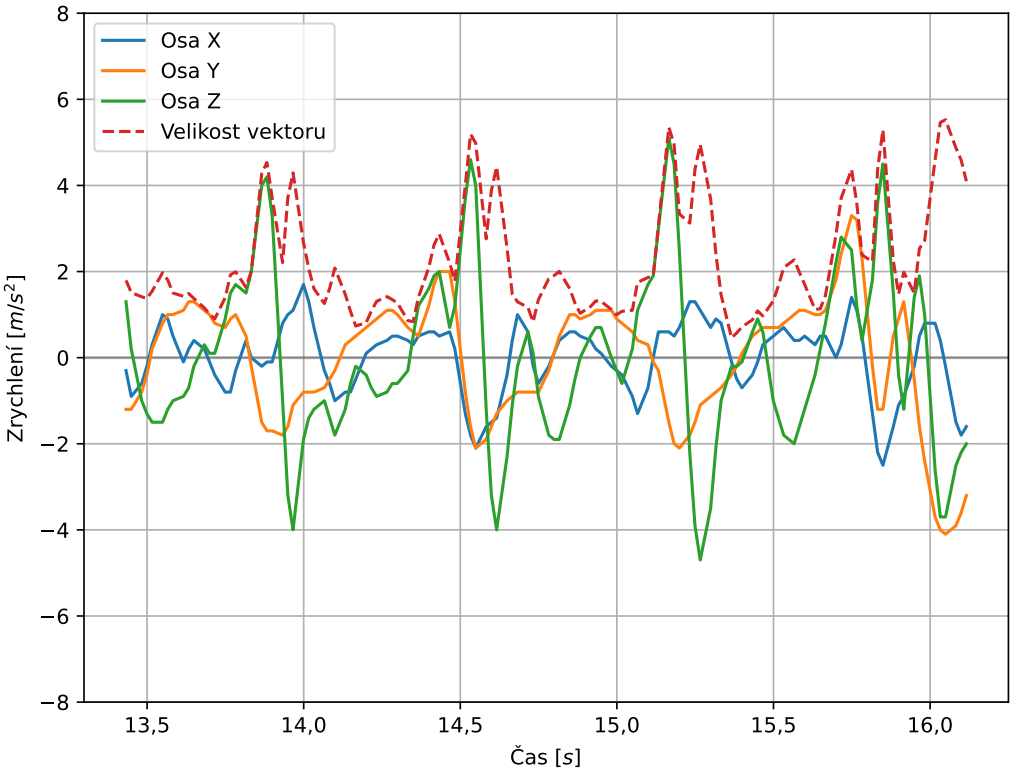
4.4.1 Akcelerometr

Pro získání zrychlení zařízení se používá akcelerometr. Ten však ukazuje veškeré zrychlení, kterému zařízení podléhá – gravitační a lineární. Proto nabízí Generic Sensor API dva další senzory, které tyto složky separují – senzor lineárního zrychlení a gravitační senzor. Budou prozkoumány tyto separované hodnoty, tedy data senzoru lineárního zrychlení a senzoru gravitace.

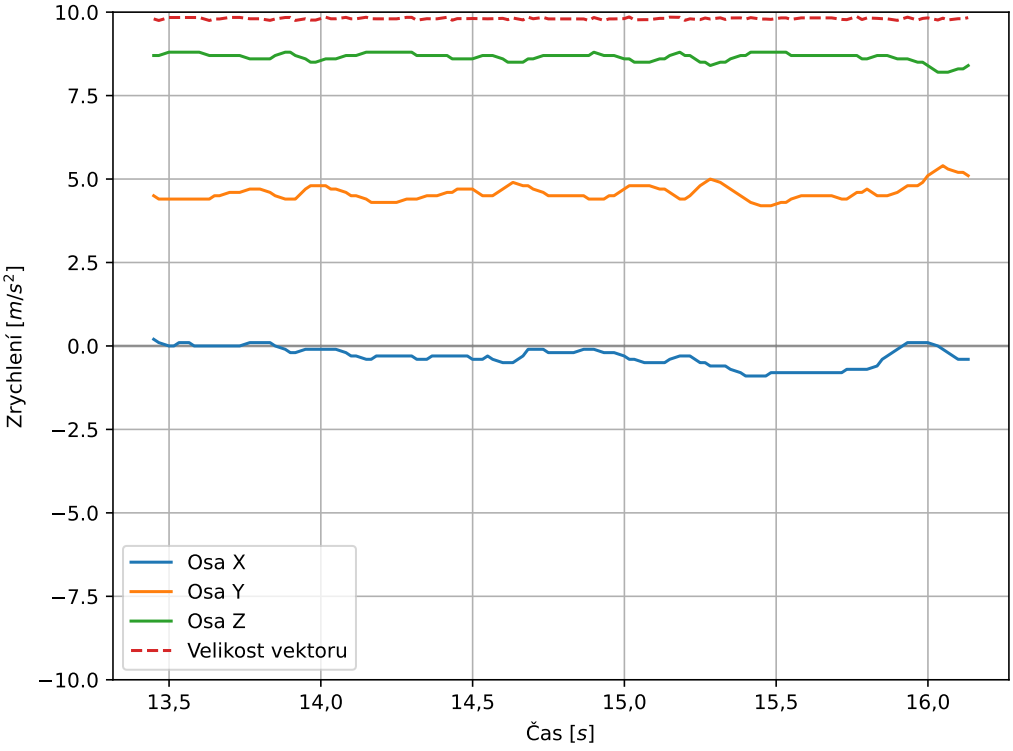
Na obrázku 4.4 lze vidět část průběhu senzoru lineárního zrychlení z dat získaných při chůzi. Svislá osa značí zrychlení zařízení v daném směru v čase uvedeném na horizontální ose. Osy pohybu zařízení jsou však relativní k jeho orientaci. Součástí grafu je také celková velikost výsledného vektoru, která je zobrazena jako přerušovaná červená čára.

V grafu jsou na první pohled vidět momenty zvýšené aktivity. Těmito momenty jsou nejspíše došlápnutí. Dále je vidět, že nejsilnější složkou zrychlení je osa Z. Osa Z je kolmá k obrazovce, zařízení tedy bylo pravděpodobně drženo v orientaci, při které obrazovka směřovala převážně vzhůru. Když byla data prozkoumána v prostorové aplikaci, popsané v kapitole 4.3, tato orientace se potvrdila.

[1] Identifikátor těchto dat je *842c2cab-6179-4c91-bf03-75544f6f35a0*



Obrázek 4.4: Průběh hodnot senzoru lineárního zrychlení při chůzi



Obrázek 4.5: Průběh hodnot senzoru gravitace při chůzi

Na obrázku 4.5 lze vidět, že velikost vektoru gravitace (tj. $\sqrt{x^2 + y^2 + z^2}$) zůstává téměř konstantní. Chyba nastává v důsledku předem zmíněného nízkého rozlišení senzorů zrychlení. Hodnoty na různých osách se mění se změnou orientace zařízení.

4.4.2 Orientace

Orientaci zařízení popisují dva senzory – senzor relativní orientace a senzor absolutní orientace. Absolutní orientace zohledňuje souřadnicový systém Země. Budeme prozkoumávat hodnoty senzoru relativní orientace, která nám udává orientaci zařízení relativní k jeho umístění na Zemi. Dále gyroskop popisuje současnou úhlovou rychlost zařízení okolo jednotlivých os. Hodnoty gyroskopu by měly korespondovat se změnami hodnot senzorů orientace.

Obrázek 4.6 ukazuje průběh rychlostí otáčení se zařízením získaných z gyroskopu. Vertikální osa zde zobrazuje úhlovou rychlost kolem dané osy zařízení. Na rozdíl od senzorů akcelerace je rozlišení dat mnohonásobně vyšší, nejmenší naměřený rozdíl mezi různými hodnotami dosáhl $5,83\text{e}{-9}$.

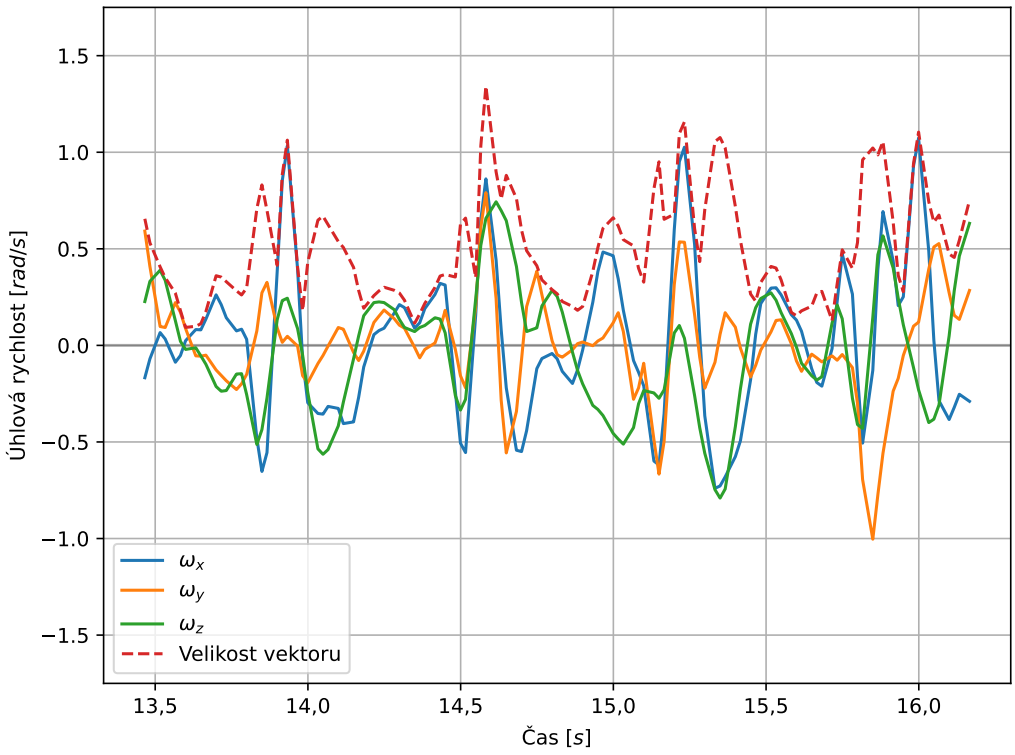
Na obrázku 4.7 lze vidět naměřená data orientace. Hodnoty na vertikální ose popisují kvaternion rotace v daném okamžiku. Součet čtverců těchto hodnot musí vycházet 1. Stejně jako u předchozích senzorů je možné vypozorovat momenty došlapu při chůzi, ale změna hodnot je mnohem jemnější. V porovnání s gyroskopem je hlavním rozdílem možnost vyčtení současné orientace zařízení bez potřeby znát jeho předchozí stav.

4.4.3 Časové značky

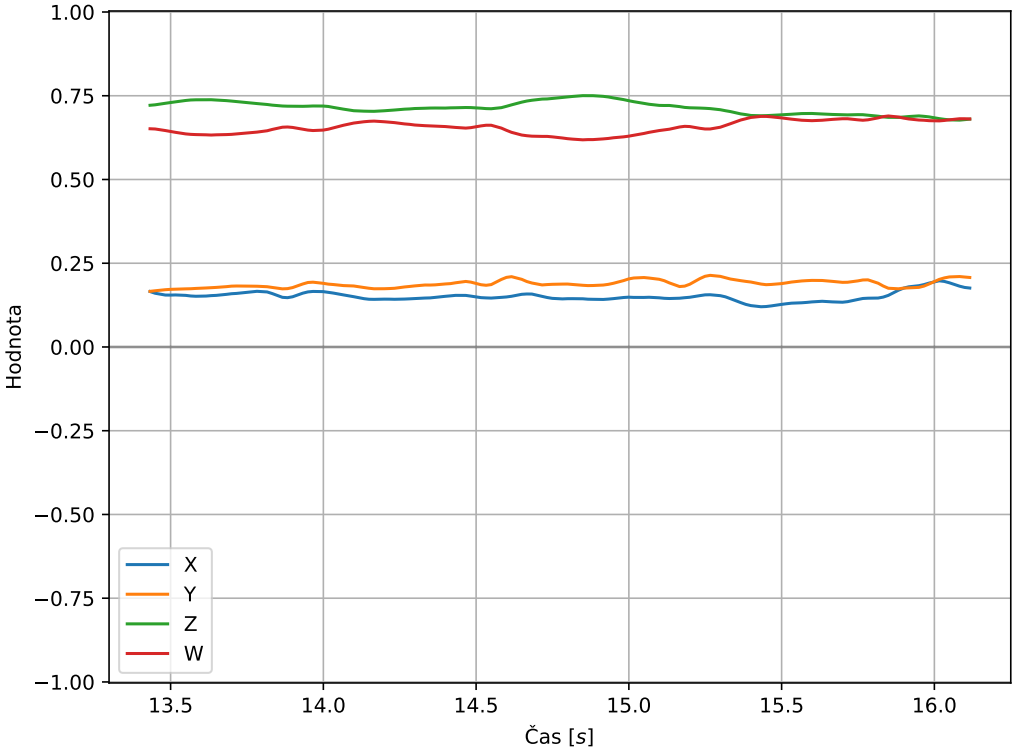
Pro ukázku odlišnosti v jednotlivých mobilních zařízeních a jejich konfigurací se lze podívat na poskytovanou rychlost čtení jejich senzorů. Obrázek 4.8 ukazuje kumulativní podíl záznamů seřazených podle prodlevy od posledního záznamu. Vertikální osa reprezentuje podíl záznamů, které měly prodlevu od posledního záznamu stejnou nebo nižší, než je současná hodnota na horizontální ose.

Na grafu lze vidět různá mobilní zařízení, na kterých byla sbírána data. Ukázány jsou také dva různé sběry na zařízení *Samsung Galaxy S21 FE*, které od sebe byly vzdáleny necelé dva měsíce. Záznamy se na grafu značně liší, což ukazuje, že rychlost čtení závisí nejen na zařízení, ale také na jeho současné konfiguraci. Důvodů pro změnu může být několik, např. jiná verze operačního systému či prohlížeče nebo různé zatížení hardware.

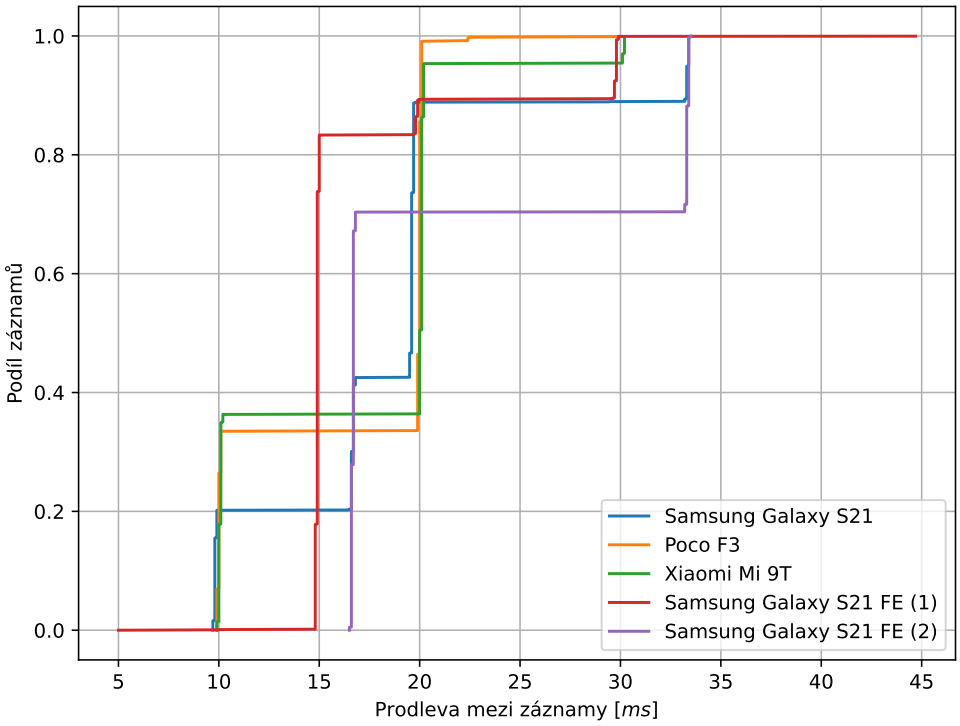
Obrázek 4.9 pak ukazuje stejnou statistiku jako minulý graf, ale pro jednotlivé senzory na zařízení. Lze tedy vidět, že i v rámci typů senzorů se může rychlost čtení lišit.



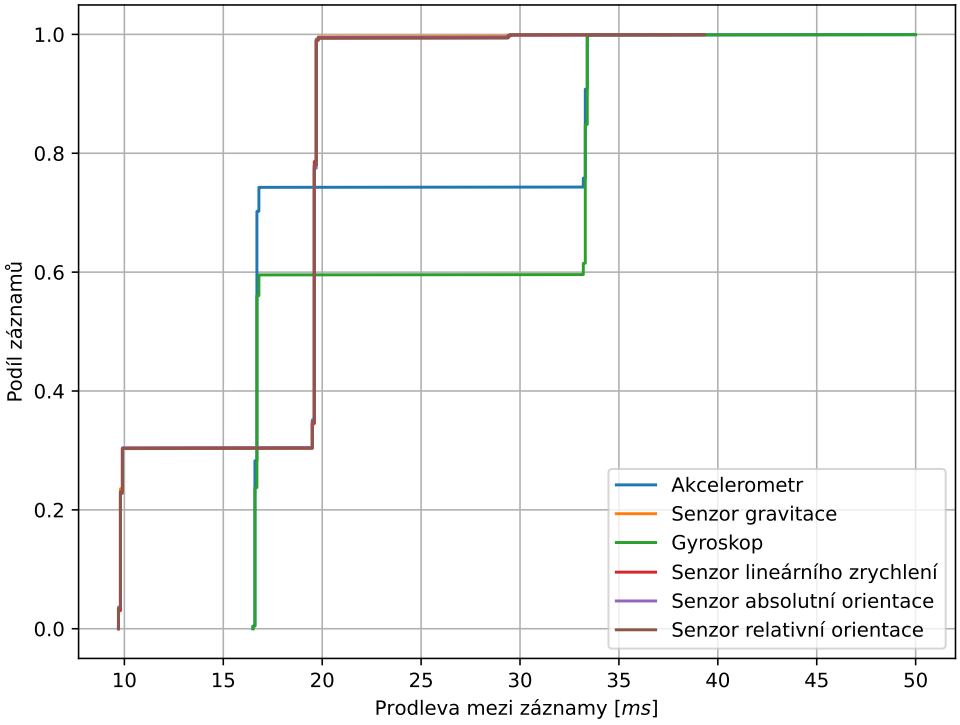
Obrázek 4.6: Průběh hodnot gyroskopu při chůzi



Obrázek 4.7: Průběh hodnot senzoru relativní orientace při chůzi



Obrázek 4.8: Kumulativní podíl záznamů pro různá mobilní zařízení podle časové prodlevy od posledního záznamu



Obrázek 4.9: Kumulativní podíl záznamů pro různé senzory podle časové prodlevy od posledního záznamu

Kapitola 5

Návrh řešení

V této kapitole bude popsán postup návrhu řešení. První část popisuje požadavky na simulaci pohybu. Dále je popsán zvolený způsob simulace, kterým je kinematická simulace kostry s vychýlením kostí pomocí sinusoid. Následuje návrh získávání vhodných parametrů těchto sinusoid, způsobu interpolace mezi různými parametry a vizualizace generovaného pohybu. V poslední části jsou uvedena možná rozšíření navrženého řešení.

5.1 Požadavky simulace

Požadavky na způsob simulace se odvíjejí z integrace výsledného řešení do rozšíření JShelter.

Uvěřitelné hodnoty

V současném stavu rozšíření JShelter jsou hodnoty senzorů generovány dle náhodně zvolené statické orientace. Pokud by senzory byly zkoumány, lze jednoduše dojít k závěru, že se senzory bylo manipulováno. Výsledkem navržené simulace by tedy měly být hodnoty, které se v průběhu času mění uvěřitelným způsobem, který následuje způsob pohybu člověka.

Výpočetní nenáročnost

Senzory zařízení jsou schopny poskytovat data s velmi velkou frekvencí. Implementovaná simulace by tedy měla také být schopna poskytovat hodnoty ve stejné frekvenci, a to s co nejmenšími požadavky na výkon.

Bezestavový výpočet

V rozšíření JShelter je kladen důraz na poskytování stejných dat stránkám na stejné doméně. V současné stacionární verzi simulace je pro výběr orientace zařízení použito pseudonáhodné číslo vygenerované podle hash hodnoty domény.

Nově implementovaná simulace bude tento princip následovat. Simulace tedy musí být schopna generovat stejná data v různých kontextech bez vědomí o předchozím možném stavu simulace.

Neopakující se výsledky

Analýzou generovaných dat v přiměřeném časovém úseku by nemělo být možné detekovat sérii přímo se opakujících vygenerovaných dat.

Způsoby pohybu

Řešení by mělo být schopno simulovat velké množství různých pohybů člověka, který zařízení drží. Mezi hlavní pohyby patří:

- chůze,
- běh,
- stání,
- sezení.

Dále by však byla žádoucí simulace alternativních stavů, jako např. stacionární zařízení na rovné ploše, či uživatel ve vozidle.

Změna stavu

Při dlouhém používání určité webové stránky či aplikace není realistické, že by uživatel prováděl stále stejný pohyb (např. 10 minut chůze). Simulaci by tedy mělo být možné pozvolně převést na jiný způsob pohybu.

Rychlost čtení

V sekci 4.4.3 bylo ukázáno, že senzory mohou poskytovat data v různých časových intervalech, záležících na modelu zařízení, jeho konfiguraci i typu samotného senzoru. Simulace by tedy měla také být schopna poskytovat uvěřitelná data v libovolných časových intervalech.

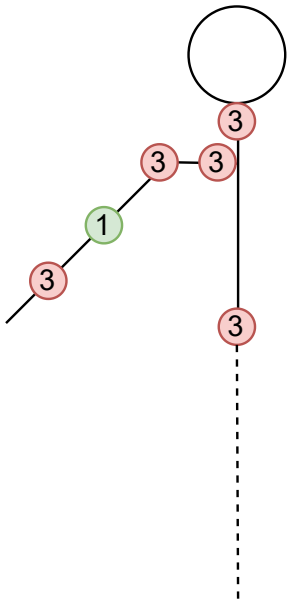
5.2 Model kostry

Řešení bude využívat kostry člověka, popsané v části 3.1. Kostra bude dále zjednodušena podle požadavků na tuto simulaci a na jednotlivé klouby budou aplikována omezení pro zvýšení pravděpodobnosti vytvoření uvěřitelného pohybu. Bude omezeno maximální vychýlení po jednotlivých osách. Omezení nebude aplikováno přímým omezením hodnoty, ale pozvolným zmírněním vychýlení při přibližování nastaveným limitům. Následkem toho bude realistické postupné zpomalení pohybu.

V navrhovaném modelu bude vlastní osou pro každou kost osa Y. Pokud je tedy potřeba vytvořit kloub, který má pouze 1 stupeň volnosti okolo osy X (např. loket), je třeba nastavit maximální vychýlení okolo os Y a Z nulové. Jakýkoli pohyb je poté na kloub aplikovaný, dojde k náklonu pouze kolem osy X.

Zjednodušení bude spočívat především v odstranění nepotřebných částí těla simulované kostry. Odstraněna bude ruka nedržící zařízení, jelikož výsledná simulace bude kinematická a pohyb druhé ruky tedy nebude nijak ovlivňovat pohyb simulovaného zařízení. Dále nebudou simulovány spodní končetiny, jejich pohyb bude aproximován jednodušší simulací.

Návrh kostry lze vidět na obrázku 5.1. Lze vidět, že kostra obsahuje také hlavu. Ve výsledném řešení se hlava nebude objevovat, pro tvorbu modelu je však žádoucí znát polohu hlavy a díky tomu polohu očí. Koncovými efektory budou v této kostře tedy dlaň (pro pozici zařízení) a hlava (pro pozici očí).



Obrázek 5.1: Návrh kostry pro řešení se zobrazenými klouby a jejich stupni volnosti

5.3 Generace pohybu

Výsledná poloha simulovaného zařízení se bude skládat ze dvou hlavních částí – složený pohyb kostí a pohyb celého těla.

Pro generování pohybu kostí bude využito dopředné kinematiky. Hodnoty vychýlení kloubů však nebudou vytvořeny předem, ale generované za běhu. Ke generaci bude využito skládání sinusoid s parametry, které mají za cíl vytvořit realistický pohyb. Pro pohyb modelu kostry bude využita manipulace rotací kloubů. Rotace kloubů bude specifikována 3 hodnotami:

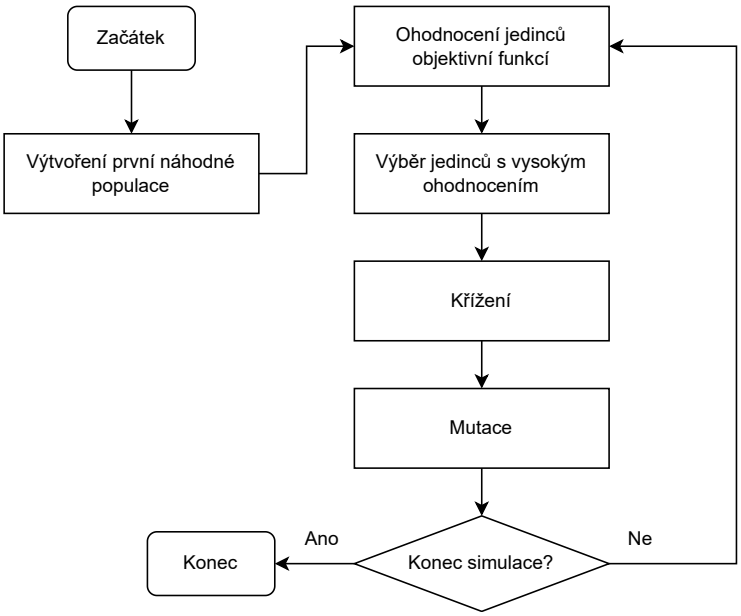
- *angle* - úhel, který popisuje směr vychýlení po rovině kolmé k ose připevněné kosti,
- *amount* - velikost vychýlení ve směru daném hodnotou *angle*,
- *roll* - popisuje náklon kloubu, neboli otočení připevněné kosti kolem vlastní osy.

Omezení pro vychýlení kloubů popsané v sekci 5.2 bude provedeno aplikováním sigmoidy na hodnotu *roll* a na výsledné vychýlení kolem os X a Z získané výpočtem z hodnot *angle* a *amount*. Funkce sigmoidy (s rozsahem od -1 do 1) bude mít následující tvar:

$$sigmoid(x) = 2 * \frac{e^x}{1 + e^x} - 1$$

Každý kloub bude mít konstantní složku a složku generovanou skládáním sinusoid. Sinusoidy pro generování hodnot *angle*, *amount* a *roll* budou mít 3 parametry – frekvence, fázový posun a amplituda. Výsledná hodnota (označme x) v čase t s konstantní složkou C a n sinusoidy s parametry A (amplituda), F (frekvence) a P (fázový posun) tedy bude získána následovně:

$$x(t) = \sum_{i=1}^{n} (A_i * \sin(F_i * t + P_i)) + C$$



Obrázek 5.2: Obecný průběh genetického algoritmu

Pohyb celého těla bude simulovaný podobným způsobem jako pohyb kloubů. Tělo bude mít určitou konstantní rychlost, která bude určena typem pohybu a použitým modelem. Dále bude tělo vychýleno horizontálně a vertikálně pomocí série sinusoid, jejichž efektem by měl být periodický pohyb připomínající lidskou chůzi. Horizontální úhel pohybu bude také generovaný pro umožnění jiného než přímého pohybu.

5.4 Nalezení parametrů

Pro nalezení parametrů pohybu kloubů a těla, popsaných v předešlé části 5.3, bude využito genetického algoritmu. Genetické algoritmy se využívají pro nalezení přibližného globálního maxima v optimalizačních úlohách. Jsou založeny na konceptech přirozeného výběru a dědičnosti a vhodné pro problémy, pro které existuje více možných řešení. Pro každý krok genetického algoritmu je získáno ohodnocení jedince pomocí objektivní funkce, které definuje daný problém (selekce). Jedinci, kteří získali vysoké ohodnocení, mají vyšší šanci rozmnožení (křížení dvou jedinců), po kterém následuje mutace potomků. [13]

Pro získání dobrých výsledků v rozumném čase je tedy důležité definovat vhodnou objektivní funkci pro hodnocení získaných parametrů. Jako metriky je možné využít některé charakteristiky nasbíraných dat nebo definovat chtěné vlastnosti výsledné simulace. Pro získání dobré objektivní funkce bude třeba experimentovat s různými způsoby vyhodnocení modelu. Možnými požadavky jsou např. následující:

- Obrazovka zařízení by měla být čitelná uživatelem, tj. jeho oči by měly na obrazovku vidět.
- Výchylky zrychlení a orientace simulovaného zařízení by se měly blížit nasbíraným datům.
- Frekvence velkých zákmitů v získaných datech by měla korespondovat s frekvencí chůze.

Objektivní funkce bude vyhodnocovat tyto metriky pro určité časové úseky. Nejprve se podívá na delší časový úsek (např. 5 s) počínaje od začátku simulace, tedy od času 0. V daném úseku bude provedena simulace s podobnou frekvencí jako senzory zařízení, tj. 100 Hz neboli pro každých 10 ms. Dále bude vybráno několik kratších časových úseků v různých dalších časech, které budou kontrolovat, zda je simulace stabilní.

Jelikož kroků simulace bude pro každé ohodnocení provedeno velké množství, pro urychlení nalezení parametrů bude žádoucí funkci ohodnocení vytvořit s velkým ohledem na výkon, např. v nízkoúrovňovém jazyce nebo s použitím optimalizovaných knihoven.

5.5 Změna pohybu

Jak bylo popsáno v požadavcích na simulaci v části 5.1, je důležitou možností mezi různými pohyby přepínat za běhu simulace. Výhodou tohoto návrhu je velmi jednoduchá interpolace mezi dvěma různými pohyby. Pokud bychom chtěli plynule přejít z pohybu generovaného funkcí $x_A(t)$ do pohybu s funkcí $x_B(t)$ počínaje v čase t_1 s trváním Δ_t, lze toho docílit např. jednoduchou lineární interpolací:

$$x(t) = \frac{(t_1 + \Delta_t) - t}{\Delta_t} * x_A(t) + \frac{t - t_1}{\Delta_t} * x_B(t)$$

Je však možné využít libovolnou funkci pro interpolaci dvou hodnot. Výsledkem by měl být plynulý přechod bez nutnosti specifikace dodatečných přechodových stavů.

5.6 Vizualizace

Pro kontrolu vygenerovaných modelů simulace bude rozšířen nástroj ze sekce 4.3. Bude umožněno zobrazení kostry a jejích pohybů a ohodnocení daného modelu objektivní funkcí. Bude tedy možné ověřit, zda je zvolená objektivní funkce správná pro daný cíl a zda generovaný pohyb vypadá vizuálně dostatečně realisticky. Také bude umožněno mezi různými modely pohybu interpolovat a ověřit tak správnost strategie pro změnu pohybu.

Důležitým faktorem je ověření, zda funkce ohodnocení využívaná pro genetický algoritmus pracuje identicky jako tato vizualizace. Pokud by tomu tak nebylo, mohlo by se stát, že správně získané parametry z genetického algoritmu budou ve vizualizaci vydávat špatné výsledky.

5.7 Možná rozšíření návrhu

V této části bude uvedeno několik možných rozšíření, která by mohla simulaci vylepšit. Uvedená rozšíření budou prozkoumána, ale jejich implementace není zásadní složkou výsledného řešení.

Dynamická simulace pohybu těla

V současném návrhu je pohyb celého těla generovaný stejným způsobem jako pohyb kostí, jehož následkem může být nerealistická charakteristika chůze. Pokud by bylo možné použít dynamickou simulaci, např. model inverzního kyvadla popsaného v části 3.3.2, mohla by se uvěřitelnost simulace zvýšit. Takový způsob simulace však může narušit požadavky na výpočetní nenáročnost a bezestavový výpočet.

Přechodové stavy

Přestože mezi různými pohyby jde efektivně interpolovat, vytvořením přechodových stavů (např. sedání si na židli, zvedání se ze židle, začátek chůze, konec chůze atd.) by se mohla uvěřitelnost simulace v těchto situacích zvýšit. Přechodových stavů by však mohlo být velmi velké množství a musela by být definována pravidla pro přechody mezi nimi.

Komplexní kostra

Kostru modelu by bylo možné vytvořit více komplexní – definovat další vztahy mezi kostmi, simulovat svaly a definovat další omezení, která by mohla vyústit v realističtější pohyb. Nebezpečím tohoto přístupu je příliš velká výpočetní náročnost, a kvůli tomu také prodloužení doby potřebné pro nalezení vhodných parametrů.

Kapitola 6

Implementace

Tato kapitola popisuje postup implementace simulace pohybu zařízení, která byla navržena v kapitole 5. Řešení se dělí na 3 hlavní části, jejichž pořadí implementace bylo zvoleno kvůli závislostem a možnostem ověření jejich korektnosti:

1. tvorba kostry, implementace jejího pohybu a vizualizace,
2. generace parametrů kostry pro lidský pohyb,
3. integrace řešení do webového rozšíření JShelter.

6.1 Tvorba kostry a vizualizace

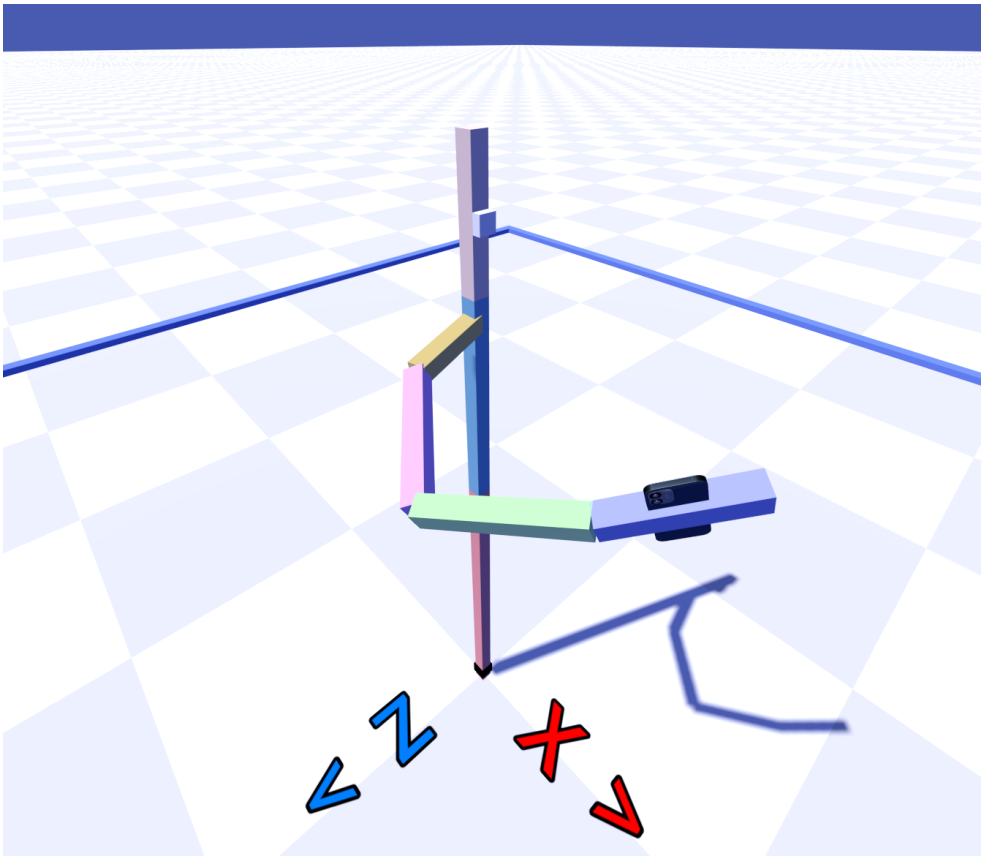
Tato část implementace byla integrována do aplikace pro zobrazení prostorových dat, která je představena v podkapitole 4.3. Pro vizualizaci byl použit herní engine Godot a jazyk C#. Přestože Godot poskytuje vlastní třídy pro 3D struktury a transformace, bylo využito generického řešení poskytovaného v jazyce C# (*System.Numerics*), jmenovitě tříd *Vector3*, *Quaternion* a *Matrix4x4*. Výhodou tohoto postupu je menší závislost kódu na herním enginu, a tedy jeho lepší přenositelnost.

Výsledkem této části je aplikace, která nám umožní zobrazit pohyb zařízení i celé kostry podle předaných parametrů pohybu. Použita bude k ověření správnosti chování simulace a k ohodnocení parametrů simulace, jejichž generace je popsána v následující podkapitole.

6.1.1 Kostra

Kostrou je objekt, který obsahuje a řídí jednotlivé kosti a další části modelu. Součásti modelu kostry jsou následující:

- Nohy,
- Kosti podílející se na pohybu simulovaného zařízení (trup, rameno, horní a spodní paže a dlaň),
- Kost hlavy, která slouží ke generaci parametrů modelu,
- Koncové efektory (oči a zařízení).



Obrázek 6.1: Vizualizace kostry v klidu

Každá kost je vykreslena jako tenký kvádr. Kostem jsou přiděleny náhodné barvy, aby byly jednodušeji rozeznatelné. Kost může být připevněna k další kosti nebo k nohám (společným názvem je *SkeletonPart*) a způsob připevnění dále určují hodnoty odsazení od konce (*AttachedOffset*) a rotace (*AttachedRotation*). Při hodnotě odsazení 0 přiléhá kost ke konci předchozí kosti a při hodnotě 1 přiléhá k začátku předchozí kosti. Rotace je definovaná eulerovskými úhly a umožňuje libovolné otočení kosti vůči kosti předchozí. Vizualizace kostry lze vidět na obrázku 6.1.

Pro určení délky jednotlivých kostí byly využity průměrné rozměry těla dospělého muže, uvedené v práci [22]. Rotace kostí vůči předchozím kostem a omezení maximálních vychýlení, které bylo popsáno v části 5.2, byly zvoleny pomocí pokusů takové, aby libovolné parametry pohybu kostí vyústily v uvěřitelnou polohu kostry. Výsledné parametry transformací a limitů kostry jsou ukázány v tabulce 6.1.

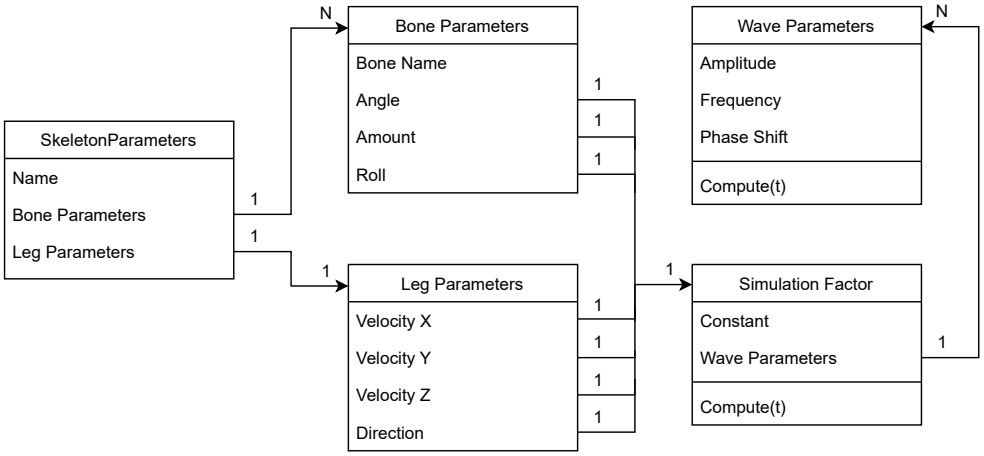
Speciální částí kostry jsou nohy, které jsou reprezentovány pouze jednou kostí a mají za úkol aproximovat celkový pohyb člověka. Jako jediná část kostry nemají předka a určují tedy počáteční polohu a rotaci celé kostry.

6.1.2 Parametry pohybu

Pohyb kostry je řízen parametry. Základem parametrů je třída *SimulationFactor*, reprezentující parametry pro výpočet jedné části pohybu, který byl definovaný v podkapitole 5.3. Tato třída faktoru má konstantní složku a seznam parametrů (amplitud, frekvencí a fázo-

Tabulka 6.1: Použité parametry částí kostry

Jméno části	Délka	Rotace vůči předchozí	Offset	Limity úhlů
Nohy	100 cm	-	-	-
Trup	55 cm	0°, 0°, 0°	0	15°, 25°, 28°
Hlava	30 cm	0°, 0°, 0°	0	0°, 0°, 0°
Rameno	20 cm	98°, 15°, 0°	0,1	17°, 0°, 0°
Paže	30 cm	15°, 10°, -35°	0	65°, 60°, 70°
Předloktí	27 cm	0°, 0°, -70°	0	0°, 0°, 70°
Dlaň	21 cm	0°, 0°, -15°	0	25°, 90°, 75°
Oči	-	0°, 0°, -90°	0,5	-
Telefon	-	0°, 90°, -90°	0,5	-



Obrázek 6.2: Diagram tříd parametrů pohybu kostry

vých posunů) pro sinusoidy. Pro výpočet výsledku faktoru v čase t byla vytvořena funkce *Compute(t)*.

Parametry pro pohyb kostí obsahují jméno cílové kosti a 3 složky (faktory) – *angle*, *amount* a *roll*. Parametry pro pohyb nohou mají zcela odlišné složky – 3 faktory, které popisují pohyb nohou po jejich 3 relativních osách (X, Y a Z), a jeden faktor který reprezentuje natočení nohou.

Seznam parametrů pro kosti a parametry pro pohyb nohou jsou sdruženy ve třídě *SimulationParameters*, která reprezentuje jednu celkovou sadu parametrů. Celkovou strukturu tříd parametrů pohybu lze vidět na obrázku 6.2. Jelikož parametry budou předávány mezi různými projekty, bylo pro jejich serializaci využito formátu *JSON*. Pro samotnou serializaci a deserializaci v prostředí .NET bylo využito balíčku *Newtonsoft.Json* kvůli lepšímu chování ve výchozím nastavení oproti vestavěnému řešení *System.Text.Json*.

6.1.3 Pohyb kostry

Simulace pohybu zařízení spočívá v postupné transformaci všech částí kostry vedoucích k zařízení. Protože transformace každé kosti je závislá na předešlé části kostry, je nutné provádět simulaci jednotlivých kostí ve správném pořadí. Výsledkem simulace kosti je zjištění její počáteční polohy a směru. Následující kosti poté mohou tyto výsledky a délku

```
public void UpdateBone(float roll, float angle, float amount)
{
    // Výpočet počátečního umístění kosti
    Matrix4x4 matrix = Matrix4x4.CreateFromQuaternion(Parent.Rotation);
    Vector3 offset = new(0, Parent.Length * (1 - AttachedOffset), 0);
    Vector3 rotatedOffset = Vector3.Transform(parentOffset, matrix);
    Location = Parent.Location + rotatedOffset;
    // Výpočet natočení kosti
    float yRot = Sigmoid(roll) * MaxAngles.Y;
    float xRot = MathF.Cos(angle) * Sigmoid(amount) * MaxAngles.X;
    float zRot = MathF.Sin(angle) * Sigmoid(amount) * MaxAngles.Z;
    Quaternion xzRotation = Quaternion.CreateFromYawPitchRoll(0, xRot, zRot);
    Quaternion yRotation = Quaternion.CreateFromYawPitchRoll(yRot, 0, 0);
    Rotation = Parent.Rotation * AttachedRotation * xzRotation * yRotation;
}
```

Výpis 6.1: Zkrácená funkce pro pohyb kosti

```
public void UpdateLegs(float delta, Vector3 velocity, float direction)
{
    Matrix4x4 matrix = Matrix4x4.CreateFromAxisAngle(Vector3.UnitY, direction);
    Vector3 rotatedVelocity = Vector3.Transform(velocity, matrix);
    Location += rotatedVelocity * delta;
    Rotation = Quaternion.CreateFromAxisAngle(Vector3.UnitY, direction);
}
```

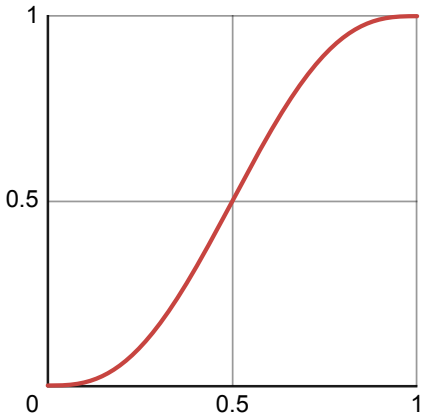
Výpis 6.2: Funkce pro pohyb nohou

dané kosti použít pro svou vlastní simulaci. Výsledkem pohybu všech kostí je zjištění poloh a rotací koncových efektorů – očí a zařízení.

Ve výpisu 6.1 lze vidět funkci, která podle předaných parametrů provede transformaci kosti. Nejprve je vypočítána počáteční pozice kosti podle pozice, rotace a délky předka a parametrů přichycení. Poté je vypočítána rotace kosti, která závisí na rotaci předka, parametrech přichycení a předaných parametrech *angle*, *amount* a *roll*. Účinek předaných parametrů je omezen funkcí *Sigmoid* popsané v podkapitole 5.3 a určeným maximálním vychýlením dané kosti v jednotlivých osách.

Důležitou vlastností této funkce je fakt, že její výsledek (stav proměnných *Location* a *Rotation* po jejím provedení) je závislý pouze na předaných parametrech a stavu předchozí části kostry. Tento bezestavový výpočet umožňuje započetí simulace v libovolném čase pohybu a pro stejný předaný čas znamená stejné výsledky.

Výpis 6.2 ukazuje postup transformace nohou. Na rozdíl od ostatních kostí nohy nejsou bezestavové. Udržují si své poslední umístění, ke kterému poté přičítají další posun. Jelikož výsledná implementace v rozšíření JShelter nebude poskytovat přístup k absolutní poloze, ale pouze ke zrychlení simulovaného zařízení, není využívání poslední polohy negativní vlastností.



Obrázek 6.3: Průběh funkce *smootherstep* v rozmezí vstupní hodnoty 0 až 1

Parametry pohybu je možné interpolovat a díky tomu plynule přecházet z jednoho typu pohybu do druhého. Při použití lineárního přechodu, ukázaného v části 5.5, byla změna pohybu velmi nárazová a nepůsobila přirozeně. Pro přechod bylo tedy využito funkce pro zlehčení přechodu *smoothstep*, přesněji její variace *smootherstep*, jejíž průběh lze vidět na obrázku 6.3. K přechodu je potřeba znát počáteční a konečné parametry, čas započetí přechodu a čas konce přechodu.

6.2 Generátor parametrů kostry

Pro generaci parametrů kostry, které byly popsány v předešlé kapitole, byl vyvinut samostatný program. Cílem tohoto programu je vytvářet parametry, které povedou k pohybu kostry podobnému pohybu člověka. Je možné vytvářet různé typy pohybu a každá sada parametrů je odlišná.

Pro generaci parametrů je využito genetického algoritmu. Při vývoji bylo nejprve využíváno knihovny *PyGAD*[1] v jazyce Python. Kvůli problémům s výkonem, které by bylo obtížné vyřešit, a výhodám sdílení stejného kódu se však vývoj později přesunul na knihovnu *GeneticSharp*[2] a jazyk C#, tedy stejnou platformu jako aplikace pro vizualizaci dat z části 6.1.

Pomocí této aplikace bylo vygenerováno 40 sad parametrů pro simulaci chůze, 40 sad parametrů pro simulaci sezení a 20 sad parametrů pro simulaci zařízení položeného na stole.

6.2.1 Genetický algoritmus

V porovnání s výše zmíněnou knihovnou PyGAD vyžaduje GeneticSharp více práce pro prvotní použití. Při vytváření genetického algoritmu je nutné definovat následující položky, které implementují vyžadované rozhraní:

- Chromozom (*IChromosome*), který reprezentuje seznam parametrů (genů), definující řešení daného problému,
- Mutaci (*IMutation*), která určuje způsob náhodné změny genů v každé iteraci,

[1] https://github.com/ahmedfgad/GeneticAlgorithmPython
[2] https://github.com/giacomelli/GeneticSharp

- Fitness funkci (*IFitness*), která hodnotí výkon jedinců a jejíž výsledek ovlivní, kteří jedinci budou vybráni na křížení či postup do další generace,
- Výběr (*ISelection*), který určuje, jakým způsobem budou vybíráni jedinci pro křížení či postup v každé generaci,
- Křížení (*ICrossover*), které definuje, jakým způsobem bude mezi vybranými jedinci docházet k záměně genů,
- Ukončení (*ITermination*), které řídí, kdy se má genetický algoritmus zastavit a prohlásit za dokončený.

GeneticSharp poskytuje širokou škálu předem vytvořených tříd, které implementují vyžadované rozhraní. Příkladem těchto tříd jsou různé způsoby křížení (*UniformCrossover*, *OnePointCrossover* a další) nebo výběru (*EliteSelection*, *RouletteWheelSelection* a další). Jelikož použitá fitness funkce bude výpočetně náročná, je z časového hlediska výhodné využít možnosti paralelního spouštění úloh, poskytované třídou *ParallelTaskExecutor*.

Pro výběr je využito způsobu *EliteSelection*, který vybere z populace určitý počet nejlépe ohodnocených jedinců a dále přenáší pevně nastavený počet nejlepších jedinců beze změny. Pro křížení je využito *UniformCrossover*, který vytváří potomka z genů obou rodičů podle předaného poměru, tedy např. při poměru 0.3 bude potomek obsahovat 30% genů z jednoho rodiče a 70% genů z druhého rodiče.

Chromozom je v tomto případě nutné vytvořit vlastní. Hlavními úkoly chromozomu je vytvořit počáteční stav genů a reprezentovat v algoritmu jedince. V tomto případě je jedním genem jeden parametr simulace (např. amplituda některé ze sinusoid). Počet genů v chromozomu se tedy může měnit podle typu pohybu. Je nutné buď vytvořit různé třídy chromozomu pro různé pohyby, nebo umožnit třídě chromozomu změnit své chování podle zvoleného typu pohybu.

Pro úpravu chování chromozomu (a v následující kapitole také fitness funkce) podle typu pohybu bylo vytvořeno rozhraní *IMovementTemplate*, které má následující vlastnosti a funkce:

- *ChromosomeLength* určující kolik genů daný typ pohybu vyžaduje,
- *GetInitialGenes()* pro získání prvotních hodnot genů,
- *GenesToParameters(genes)* pro převedení genů (v tomto případě jsou geny polem hodnot typu *float*) na objekt *SimulationParameters* z části 6.1.2,
- *EvaluateSimulationResults(results)* pro ohodnocení výsledků simulace určitých parametrů,
- *SimulationStep* a *SimulationLength* pro určení délky a jemnosti simulovaného času.

Objektu chromozomu je při vytvoření předán objekt implementující toto rozhraní, který reprezentuje určitý typ pohybu. Chromozom se poté odkazuje na vlastnosti a funkce předané šablony pro správné chování. Vytvořený objekt prvotního chromozomu (nazývaný *adamChromosome*) se poté použije při vytváření objektu populace, která také definuje minimální a maximální počet jedinců v každé generaci.

Pro mutaci bylo také nutné vytvořit vlastní implementaci, jelikož žádná z výchozích nevyhovovala tomuto problému. Úkolem třídy reprezentující mutaci je implementovat funkci,

která u předaného chromozomu provede požadované změny se zohledněním předané šance na mutaci. Implementovaná mutace (s názvem *SimulationMutation*) prochází všechny geny a podle předané pravděpodobnosti určí, zda se provede či neprovede mutace genu. Samotná mutace genu je implementována jako přičtení náhodně vygenerované hodnoty k již existující hodnotě. Náhodně vygenerovaná hodnota byla nejprve rovnoměrně rozdělena v určitém rozmezí. Lepších výsledků bylo však dosaženo pomocí použití náhodných čísel s normálním rozdělením[3].

Pro způsoby ukončení trénování genetického algoritmu poskytuje GeneticSharp 4 řešení a možnost je kombinovat či vytvářet vlastní. Tyto 4 řešení jsou:

1. *GenerationNumberTermination* – ukončení po určitém počtu generací,
2. *FitnessStagnationTermination* – ukončení při stagnaci fitness hodnoty,
3. *FitnessThresholdTermination* – ukončení při dosažení určité fitness hodnoty,
4. *TimeEvolvingTermination* – ukončení podle času evoluce.

Řešení 3 nelze použít, protože není předem známá požadovaná fitness hodnota. Řešení 4 zde také není použitelné, protože doba každé evoluce zůstává přibližně stejná. Stagnace fitness hodnoty by mohla být dobrým způsobem ukončení, ale v praktickém použití byl výsledkem dlouhý čas běhu algoritmu a ne příliš dobré výsledky kvůli přetrénování výsledných parametrů. Použito bylo tedy ukončení podle počtu generací, typicky s počtem generací mezi 300 a 1000.

6.2.2 Fitness funkce

Fitness funkce je tou nejdůležitější a nejkomplikovanější částí genetického algoritmu. Cílem fitness funkce v tomto řešení je pokusit se co nejpřesněji objektivně ohodnotit pohyb vytvořený předanými parametry podle cílového typu pohybu.

Třída implementující rozhraní *IFitness* musí poskytovat funkci *Evaluate(chromosome)*, která daný stav chromozomu ohodnotí číslem typu *double*. Prvním krokem ohodnocení je simulace pohybu podle parametrů získaných z chromozomu. Délka a jemnost simulace se liší v závislosti na typu pohybu (např. pro stacionární zařízení položené na stole stačí pouze 1 vzorek, zatímco chůze simuluje 35 sekund pohybu po 20 milisekundách). V průběhu simulace jsou ukládány následující informace:

- pozice zařízení,
- rotace zařízení,
- pozice očí,
- směr nohou.

Seznamy s těmito hodnotami jsou poté použity pro vytvoření instance třídy *SimulationResults*, která reprezentuje získané výsledky ze simulace pohybu. Kromě již zmíněných seznamů hodnot jsou při vytváření třídy vypočítány také další metriky:

[3]Pro generaci náhodných čísel s normálním rozdělením bylo využito metody *Box-Muller*, jejíž implementace v jazyce C# byla získána z `https://stackoverflow.com/a/2751988/11081597`.

```
Best solution found (in 36534ms) has 28,019/37 fitness.
boneAmplitudePortions: 1,000/1
walkingAmplitudePortions: 7,889/10
averageFacingValue: 4,943/6
maxAmountValue: -0,000/-4
maxAngleValue: -0,000/-4
maxRollValue: -0,280/-4
maxLegsVelocity: -0,367/-10
...
```

Výpis 6.3: Příklad výpisu složek skóre fitness funkce

- Rychlosti a zrychlení zařízení
- Úhlové rychlosti natočení nohou a zařízení
- Natočení zařízení směrem k očím (rozmezí 0 až 1)
- Natočení zařízení směrem nahoru (rozmezí 0 až 1)
- Zarovnání roviny kolmé na osu X zařízení vůči směru nahoru (rozmezí 0 až 1)

Dále byly třídy parametrů pohybu rozšířeny o funkce, které umožňují získání dodatečných metrik přímo o parametrech. Těmito metrikami jsou teoretická maxima faktorů, poměr nenulových parametrů a poměr amplitud určitého počtu sinusoid vzhledem k celkovému součtu amplitud. Parametry pohybu a objekt s výsledky simulace jsou poté předány funkci *EvaluateSimulationResults(results)* šablony pohybu pro daný chromozom.

Při hodnocení výsledků by bylo možné přímo manipulovat s číslem reprezentujícím ohodnocení těchto výsledků, ale z důvodů jednoduššího ladění, přehlednosti a získávání informací o hodnocení byla vytvořena třída *FitnessScore*. Úkolem třídy je umožnit manipulaci se skórem (ohodnocením výsledků) pouze pomocí určitých metod, které vždy vyžadují zadání jména pro danou změnu hodnocení. Manipulovat se skórem je možné přímo (přímé přičtení či odečtení), pomocí lineárně škálované hodnoty (zadání maximální změny a hodnoty od 0 do 1 pro její škálování) či pomocí hodnoty škálované druhou odmocninou. Změna skóre může být pozitivní (zvýšení skóre) či negativní (penalizace).

Díky zadávání názvů změn skóre je možné po ohodnocení zobrazit, které z faktorů hodnocení měly jaký vliv na výsledné skóre. Příklad zobrazení lze vidět ve výpisu 6.3. Tento výpis je typicky prováděn po dokončení genetického algoritmu a kromě předem zmíněných výhod také poskytuje rychlý náhled na výsledky, což může znamenat rychlé vyřazení sady parametrů před dalšími kontrolami.

6.2.3 Příklad šablony chůze

Lidská chůze je velmi komplexní pohyb a tvorbě její šablony bylo věnováno nejvíce času. Při tvorbě instance šablony *WalkingTemplate* je nutné předat dvě hodnoty – konstantní rychlost chůze a délku trvání jednoho kroku. Tyto hodnoty jsou náhodně generované v typických rozmezích, 1,1 až 1,5 m/s pro rychlost chůze a 0,5 až 0,6 sekund pro trvání kroku. Předané

hodnoty jsou poté použity při tvorbě parametrů. Konstantní složka *VelocityX* v parametrech nohou je přesně nastavena podle předané rychlosti chůze a konstantní složky *VelocityY* a *VelocityZ* jsou nastaveny na nulu. Délka trvání jednoho kroku je přepočítána na frekvenci kroku a pevně nastavena jako frekvence první sinusoidy v každém faktoru kostry.

Všechny šablony používají jen takový počet genů, jaký je potřebný na doplnění všech parametrů, které nejsou pevně nastaveny na určitou hodnotu. Šablona chůze používá pro každou část pohybu 3 sinusoidy. To znamená 9 sinusoid na každou kost a 12 sinusoid pro pohyb nohou. Po přičtení počtu konstantních částí faktorů a odečtení počtu pevně nastavených hodnot vychází délka chromozomu (počet genů) na 168. Počáteční stav genů byl nejprve nastaven na samé nuly, lepších výsledků však bylo dosaženo pro náhodné hodnoty v rozmezí -0,025 až 0,025. Délka simulace je nastavena na 35 sekund a jemnost simulace na 20 milisekund.

Použité metriky k ohodnocení mají 3 hlavní typy:

1. Metriky hodnotící, jak se nějaká vlastnost výsledných poloh a rotací zařízení shoduje s naměřenými daty,
2. Metriky pozitivně hodnotící žádoucí vlastnosti simulace, které nelze zjistit z naměřených dat,
3. Penalizace za nežádoucí chování simulace.

Mezi 1. typ patří např. průměrné a směrodatné odchylky zrychlení, úhlových rychlostí či natočení směrem nahoru zařízení. Cílové hodnoty byly získány z nasbíraných dat, výběrem takového měření, které vykazovalo klidnou chůzi bez prudkých pohybů a náhlých změn natočení či akcelerace. 2. typem jsou v případě chůze hodnoty natočení zařízení směrem k očím a podíl sinusoid s frekvencí kroku na celkovém pohybu. Penalizace bývaly většinou přidávány jako reakce na nežádoucí chování. Příklady jsou příliš velké nebo malé změny směru chůze a rychlosti chůze, nebo příliš velké hodnoty *angle*, *amount* nebo *roll*. Váhy jednotlivých ohodnocení se značně liší. Typicky mají penalizace vyšší váhy než pozitivní ohodnocení, jelikož se v takových případech projevuje nějaký nežádoucí typ pohybu.

6.3 Integrace do rozšíření JShelter

Prvním krokem integrace řešení do rozšíření JShelter bylo přepsání využívaných funkcí .NET System.Numerics do jazyka JavaScript. Jelikož zdrojový kód .NET je veřejně dostupný, je jednoduché docílit stejného chování. Používané funkce, které bylo nutné přepsat, jsou následující:

- sčítání a odčítání 3D vektorů,
- vynásobení 3D vektoru skalární hodnotou,
- vytvoření kvaternionu (z osy a úhlu nebo z hodnot yaw, pitch a roll),
- vytvoření rotační matice z kvaternionu (vytvořená matice je velikosti 3x3, .NET poskytuje implementaci pro matici 4x4),
- násobení a inverze kvaternionů,
- rotace 3D vektoru podle rotační matice.

```
getNextTransition(time) {
    let params, lastParams, transitionStartOffset;
    let transitionEnd = this.transitionEnd;
    do {
      transitionEnd += this.random.nextMinMax(15000, 45000);
      transitionStartOffset = this.random.nextMinMax(2500, 6000);
      lastParams = params;
      params = this.random.next();
    } while (transitionEnd <= time);
    return {
      start: transitionEnd - transitionStartOffset,
      end: transitionEnd,
      params: params,
      lastParams: lastParams,
    };
}
```

Výpis 6.4: Funkce pro získání hodnot dalšího přechodu parametrů

Dále bylo třeba přepsat a upravit samotný kód kostry a jejích součástí, konkrétně tedy byly vytvořeny třídy *Skeleton*, *SkeletonBone*, *SkeletonLegs* a *SkeletonPhone*. Jedna z hlavních změn je úprava způsobu vázání kostí na jiné části kostry. V C# implementaci je předek dané kosti uchováván jako reference na daný objekt v instanci kosti. V JavaScript implementaci kosti neuchovávají informaci o předkovi – pouze hodnoty *AttachedRotation* a *AttachedOffset*. Výsledek transformace předka je předáván jako parametr funkce *move*, která také vrací výsledek své transformace ve stejném formátu. Výsledek transformace kosti se skládá z pozice, rotace a délky kosti. Výsledkem celkové simulace kostry je poloha a rotace zařízení v určitém čase. Kostra si uchovává 3 poslední výsledky z důvodu výpočtu rychlosti a zrychlení daných veličin.

Důležitou možností kostry je prolínání mezi náhodně zvolenými pohyby, popsané v části 5.5. Je nutné zabezpečit, aby byly vybrané pohyby stejné i v různých záložkách prohlížeče na stejné doméně. Pro generaci pseudonáhodných čísel podle domény již je v původní verzi rozšíření JShelter využíváno funkce *sen_prng*, která je založena na algoritmu *Mulberry32*. Pro potřeby vytváření přechodů parametry byla vytvořena třída *SenPseudoRandom*, která používá stejný algoritmus a poskytuje stejné funkce, ale poskytuje každé její instanci vlastní *seed* hodnotu. Seed hodnota je využívána a měněna při každé generaci pseudonáhodného čísla a jejím odstíněním tedy můžeme vytvořit více nezávislých generátorů pseudonáhodných čísel. Toho je využito právě při výpočtu parametrů přechodu, který používá vlastní instanci pseudonáhodného čísla pro ujištění, že série generovaných přechodů pro danou doménu bude vždy stejná. Funkci generující parametry přechodů lze vidět ve výpisu 6.4. Hodnoty *params* a *lastParams* jsou čísla v rozmezí od 0 do 1 a určují, jaké parametry ze seznamu všech parametrů se mají použít. Pro získání samotného indexu parametrů je nutné toto číslo vynásobit celkovým počtem parametrů a zaokrouhlit.

Simulace kostry však nevrací hodnoty, které by bylo možné hned předávat jako výstup pohybových senzorů. Každý typ senzoru tedy potřebuje vlastní generátor hodnot, který z výsledků simulací kostry bude získávat hodnoty vhodné pro daný typ senzoru. Pro usnadnění práce byla vytvořena další společná třída *SensorGenerator*, ze které budou jednotlivé

další generátory dědit a která zastřešuje funkce a proměnné pro obsluhu kostry a přechody parametrů jako například funkci *getNextTransition* zmíněnou výše.

Všechny dosud zmíněné části integrace byly přidány do souboru *wrappingL-SENSOR.js*, který zastřešuje společné funkce a třídy pro různé senzory. Jelikož samotné senzory jsou obsaženy v soukromém oboru názvů, je nutné jim tyto funkce a třídy předat ve formě textových řetězců. Vygenerované sady parametrů by bylo možné číst ze souboru pomocí dostupné funkce *readFile*, ale při použití docházelo k vytváření objektů senzorů před dokončením čtení tohoto souboru, což způsobilo nedostupnost parametrů pro simulaci. Jako řešení tedy bylo zvoleno vytvoření souboru *warllingL-SENSOR-DATA.js*, který obsahuje vygenerované sady parametrů ve formátu JSON přiřazené proměnné *skeleton_parameters_json*, která je vždy dostupná při vytváření senzorů.

Akcelerometr

Kód pro obalení akcelerometru se nachází v souboru *wrappingS-SENSOR-ACCEL.js*. Jako generátor hodnot byla vytvořena třída *AccelerationGenerator*. Akcelerometr zastřešuje senzor gravitace a senzor lineárního zrychlení a jeho hodnoty lze získat sečtením hodnot těchto dvou částí. Pro výpočet zrychlení je potřeba znát 3 poslední pozice a jejich časy, pomocí kterých je možné vypočítat rychlost pohybu mezi sousedními pozicemi. Z těchto rychlostí je poté vypočítáno zrychlení. Důležitou vlastností tohoto senzoru je, že zrychlení používá souřadnicový systém relativní k zařízení, který byl popsán v části 2.1.4. Při výpočtu je tedy nutné využít také rotaci zařízení.

Gyroskop

Kód pro obalení gyroskopu se nachází v souboru *wrappingS-SENSOR-GYRO.js*. Jako generátor hodnot byla vytvořena třída *GyroscopeGenerator*. Gyroskop stejně jako akcelerometr používá souřadnicový systém relativní k zařízení. Vrácené hodnoty reprezentují úhlové rychlosti okolo jednotlivých os v jednotkách radiánů za sekundu. Pro výpočet úhlové rychlosti je potřeba znát poslední 2 rotace zařízení. Pomocí vydělení rozdílu těchto rotací rozdílem jejich času můžeme zjistit potřebné úhlové rychlosti.

Orientace

Kód pro obalení senzorů orientace se nachází v souboru *wrappingS-SENSOR-ORIENT.js*. Jako generátor hodnot byla vytvořena třída *OrientationGenerator*. Existují dva senzory orientace – relativní a absolutní. V již existující implementaci bylo pozorováno, že hodnota těchto dvou typů senzorů se může lišit až o 10°, a tato vlastnost byla zachována i pro novou implementaci. Při testování tohoto senzoru bylo objeveno, že simulace je prováděna s rozdílnou orientací od hodnot naměřených na pravém zařízení – otočena o -90° okolo osy X. Při generování hodnot orientace je tedy nutné tuto rotaci obrátit, neboli výsledek otočit o 90° okolo osy X.

Magnetometr

Kód pro obalení magnetometru se nachází v souboru *wrappingS-SENSOR-MAGNET.js*. Jako generátor hodnot byla vytvořena třída *MagnetometerGenerator*. Jelikož magnetometr není ve výchozím stavu prohlížečů povolen a jeho chování tedy nebylo prozkoumáno, bude pro jeho simulaci využito již existující řešení z rozšíření JShelter, které generuje hodnoty

magnetometru z předané rotace zařízení. Simulované rotace zařízení jsou tedy předávány jako základ pro generaci hodnot magnetometru.

Kapitola 7

Experimenty

V této kapitole budou provedeny experimenty s vytvořeným řešením. Nejprve bude ověřeno správné chování různých aspektů řešení. Dále proběhne analýza výkonu řešení, které bylo integrováno do rozšíření JShelter. Nakonec proběhne porovnání simulovaného a lidského pohybu.

Po provedení změn v rozšíření JShelter proběhlo sestavení samotného balíčku s rozšířením. Přestože rozšíření JShelter je možné používat v prohlížečích postavených na bázi Chromium (Google Chrome, Microsoft Edge a další) i Firefox, Generic Sensor API je podporovaný pouze na prohlížečích typu Chromium a testování bylo nutné provádět na některém z nich. Jelikož JShelter pouze zaobaluje objekty senzorů a využívá některé jejich výchozí chování, pro jejich správné otestování je nutné provádět testování na zařízení, které jimi disponuje, tedy na mobilním zařízení. Google Chrome pro operační systém Android nepodporuje používání rozšíření, testování tedy bylo prováděno v prohlížeči Kiwi Browser, který je založený na jádře Chromium a rozšíření podporuje.

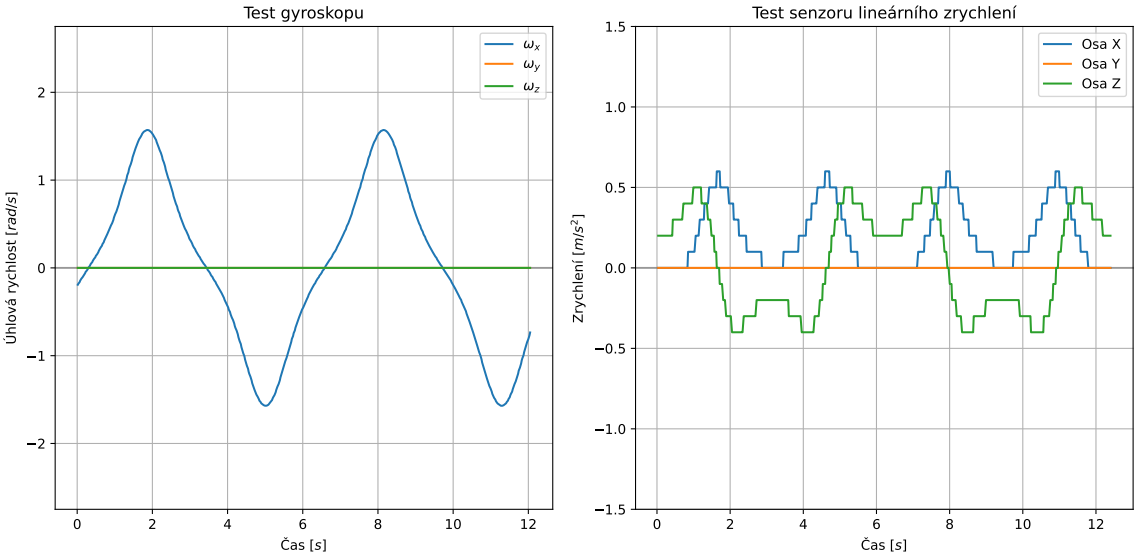
V rozšíření JShelter je implementována ochrana zranitelnosti časového razítka čtení senzorů. Tato ochrana způsobuje nepravidelné a méně časté změny hodnoty časového razítka. Pro následující experimenty byla tato ochrana vypnuta, aby bylo možné data lépe pozorovat a porovnávat.

7.1 Ověření funkčnosti

Ověření funkčnosti řešení sestává z provedení několika testů, které vyzkouší jednotlivé části simulace. Zkoušeny byly následující funkce:

- akcelerometr,
- gyroskop,
- orientace zařízení,
- přechody mezi typy pohybu.

Pro získání dat ze senzorů bude použita stejná aplikace, která byla vytvořena pro prvotní sběr dat v části 4.2. Pro ověření korektnosti dat jednotlivých senzorů byly ručně vytvořeny sady parametrů, jejichž očekávaný výsledek je předem známý. Pro akcelerometr to byla sada parametrů, při kterých se zařízení periodicky pohybovalo po své ose X (tedy displejem směrem dopředu a dozadu). Pro gyroskop to byl pohyb, při kterém se zařízení periodicky



Obrázek 7.1: Data získaná pomocí testovacích sad parametrů pro gyroskop a akcelerometr

otáčelo kolem své osy X. Pro senzor orientace byla vytvořena sada parametrů, při které bylo zařízení plně stacionární, a rotace zařízení získaná z rozšíření by se měla shodovat s rotací zobrazené v aplikaci pro vizualizaci senzorových dat.

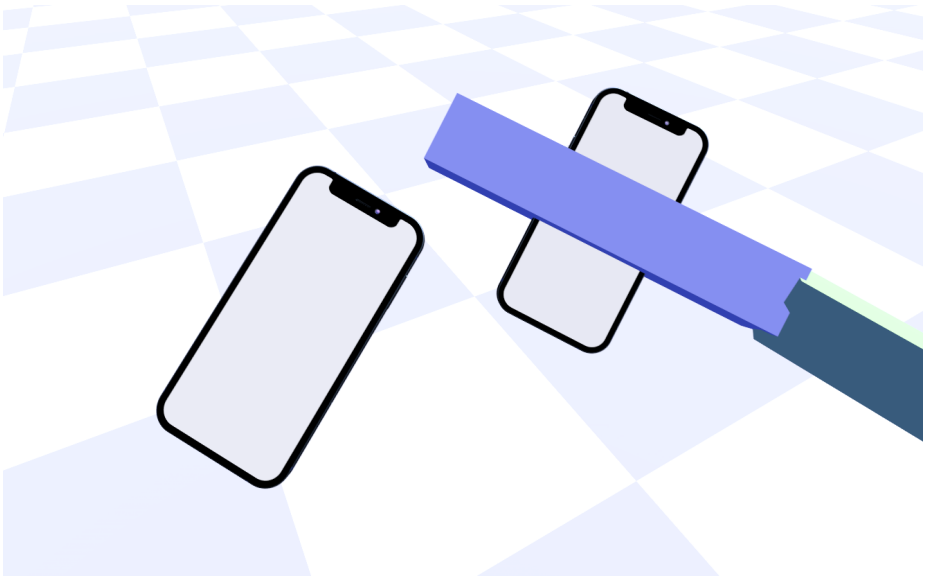
Test dat akcelerometru a test dat gyroskopu proběhly úspěšně a generované hodnoty tedy správně hlásí stav simulace zařízení. Získaná data lze vidět na obrázku 7.1. Data z gyroskopu ukazují změny pouze okolo osy X, což byl očekávaný výsledek. Data senzoru lineárního zrychlení krom očekávaných hodnot na ose Z také ukazují hodnoty v pozitivním směru na ose X. Toto je způsobeno testovací sadou parametrů, která nebyla čistě lineární, ale pro pohyb vychylovala kost předloktí (zařízení se tedy otáčelo kolem lokte kostry). Hodnoty na ose X tedy ukazují odstředivou sílu působící na zařízení. Dále lze vidět, že hodnoty akcelerometru mají malé rozlišení - to se však shoduje s daty získanými z reálných senzorů.

Test dat orientace ukázal chybu, která způsobovala rozdíl v rotaci simulace oproti hodnotám získávaných z reálných zařízení (otočení o -90° kolem osy X). Po aplikování dodatečné rotace k získané rotaci ze simulace se hodnoty shodovaly s očekávanými, což lze vidět na obrázku 7.2.

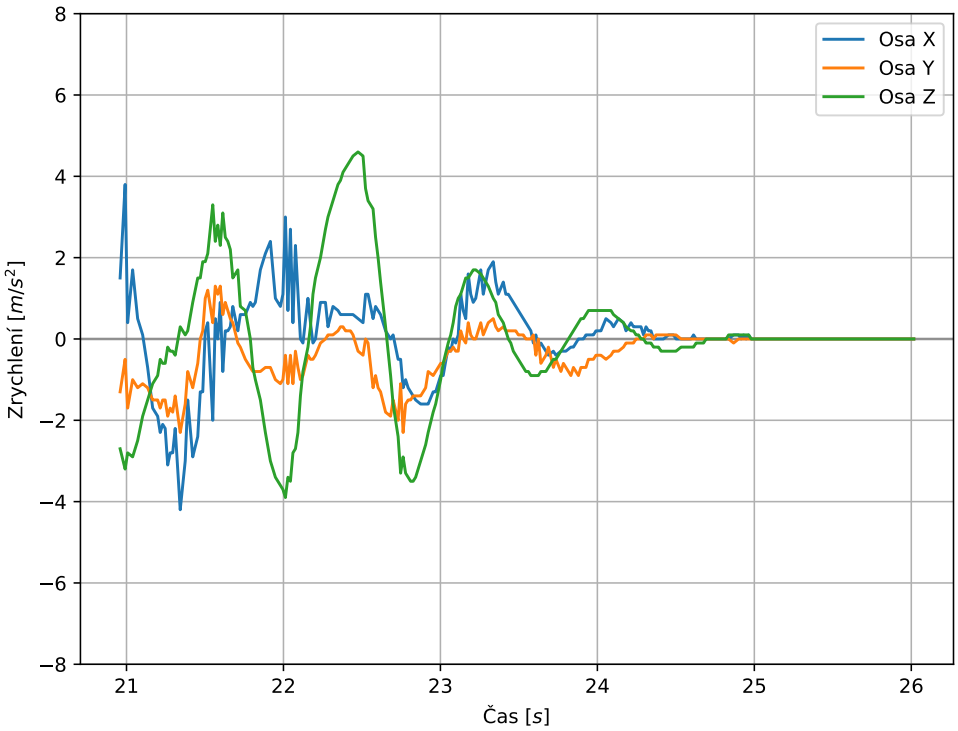
Pro otestování přechodů mezi různými pohyby byla vytvořena verze rozšíření, která obsahovala pouze 2 sady parametrů. 1. sada parametrů reprezentovala chůzi a 2. sada parametrů reprezentovala plně stacionární zařízení položené na rovné ploše. Přechod mezi těmito parametry byl tedy zřejmý a snadno pozorovatelný, jak lze vidět na obrázku 7.3. Délka vykonávání jednoho pohybu je 15 až 45 a doba prolnutí je 2,5 až 6 sekund. Data získaná z aplikace na sběr dat byla také pozorována v aplikaci pro vizualizaci dat, kde bylo zjištěno, že změna pohybu a jejich prolínání fungují tak, jak bylo očekáváno.

7.2 Výkonové testy

Jak bylo zmíněno v návrhu v sekci 5.1, jednou z důležitých vlastností řešení by měla být výpočetní nenáročnost. Kvůli způsobu, jakým JShelter mění chování senzorů, je simulace kostry prováděna pro každý aktivní senzor odděleně místo jedné společné simulace. Napří-



Obrázek 7.2: Porovnání orientace zařízení získané z testu (vlevo) s očekávanou orientací (vpravo)



Obrázek 7.3: Průběh hodnot senzoru lineárního zrychlení při přechodu mezi chůzí a stacionárním zařízením

```
function benchmark() {
  const generator = new SensorGenerator();
  const iterations = 1_000_000;
  let start = new Date();
  for (let i = 0; i < iterations; i++) {
    const time = i * 0.02;
    generator.updateTransition(time);
    generator.updateSkeleton(time);
  }
  let end = new Date();
  var time = end.getTime() - start.getTime();
  return (time / iterations) * 1000.0;
}
```

Výpis 7.1: Funkce pro měření výkonu simulace

klad aplikace pro sběr senzorových dat (popsaná v sekci 4.2), která využívá všechny typy pohybových senzorů (kromě magnetometru), způsobuje provádění 6 různých simulací kostry najednou. Pokud by tedy každý krok simulace trval příliš dlouho, následkem by byl značný dopad na výkon, a tedy i uživatelskou zkušenost.

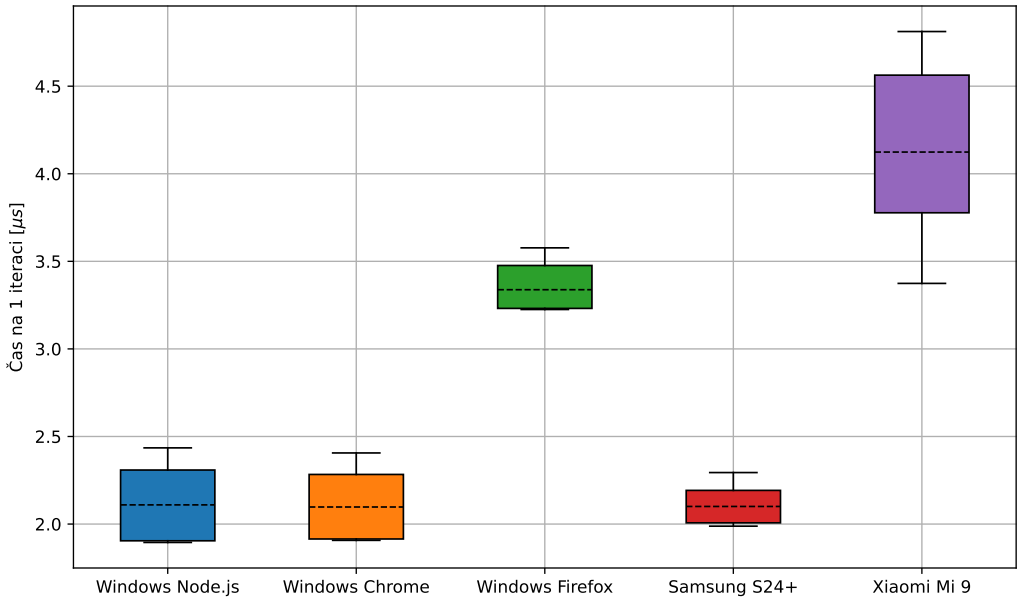
Nejdůležitější metrikou výkonu je doba potřebná pro 1 krok simulace kostry, jelikož transformace kostí obsahují velký počet matematických operací. Funkci, která byla využita pro měření výkonu, lze vidět ve výpisu 7.1. Tato funkce vrací průměrný počet mikrosekund na jednu iteraci při simulaci 1 milionu iterací. Bylo vždy provedeno 10 těchto měření a výsledky všech byly uloženy. Pro test výkonu byly zvoleny 3 scénáře:

- Stolní počítač na platformě Node.js
- Stolní počítač v prohlížeči
- Mobilní zařízení v prohlížeči

Přestože jediným realistickým scénářem je mobilní zařízení v prohlížeči, byly provedeny i další dva scénáře pro průzkum rozdílu ve výkonu na jiných platformách. Platformou stolního počítače je operační systém Windows 11, procesor AMD Ryzen 5900X, prohlížeče Google Chrome verze 124 a Mozilla Firefox verze 125 a Node.js verze 21.7. První testované mobilní zařízení je telefon Samsung Galaxy S24+ (Exynos) s operačním systémem Android 14 a prohlížečem Kiwi Browser verze 124. Druhé mobilní zařízení je telefon Xiaomi Mi 9 s operačním systémem Android 11 a prohlížečem Google Chrome verze 124.

Jak lze vidět na obrázku 7.4, průměrné časy na 1 iteraci simulace jsou velmi nízké na všech testovaných platformách – v jednotkách mikrosekund. Ve výsledcích stolního počítače lze vidět, že rychlost prostředí Node.js a V8 (prostředí pro JavaScript v prohlížeči Google Chrome) má pro tuto simulaci velmi podobný výkon. Oproti tomu je výsledek z prohlížeče Mozilla Firefox značně pomalejší. Podobná rychlost stolního počítače a telefonu Samsung Galaxy S24+ byla překvapující. Vysvětlením je velmi podobný výkon jednoho jádra[1] v obou

[1]Porovnání výsledku Geekbench 5.5 Single-Core mezi AMD Ryzen 9 5900X a Exynos 2400 z https://www.notebookcheck.net/Mobile-Processors-Benchmark-List.2436.0.html



Obrázek 7.4: Průměrné časy na 1 iteraci simulace kostry na různých platformách

procesorech, jelikož JavaScript kód je vykonáván pouze na jednom jádře. Starší telefon Xiaomi Mi 9 podává nejhorší výsledky z testovaných případů, což bylo očekávané. I v tomto nejhorším případě je však doba simulace pouze necelých 5 mikrosekund, což je výborný výsledek.

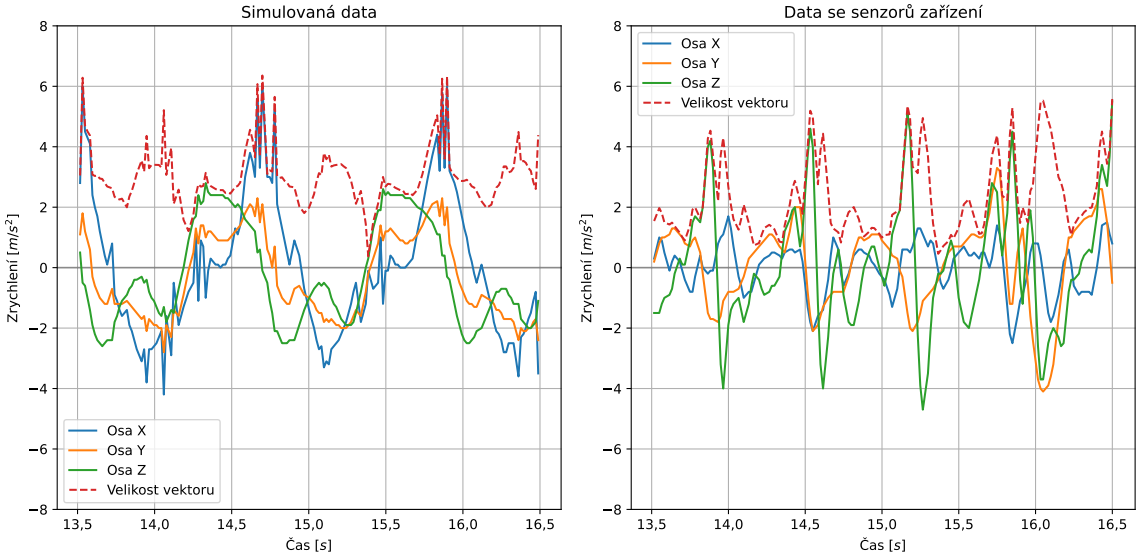
7.3 Porovnání reálných a simulovaných dat

Po ověření správného chování simulace následuje prozkoumání podobnosti generovaného pohybu k lidskému. K tomuto účelu byla porovnána data nasbíraná z mobilních senzorů s daty získanými z upraveného rozšíření JShelter. Pro porovnání byla vybrána sada parametrů pro chůzi[2].

Nejprve proběhlo vizuální porovnání prostorového zobrazení těchto dat v aplikaci ze sekce 4.3. Porovnávána byla různá měření chůze s vygenerovaným pohybem. Data vypadala velmi podobně – bylo na nich možné pozorovat podobné záchvěvy způsobené dopady nohou na zem a rotace zařízení byla uvěřitelná.

Následovalo prozkoumání samotných hodnot získaných ze simulovaných senzorů. Pro porovnání budou v následujících obrázcích zobrazena data získaná z pravých senzorů při chůzi. V obrázku 7.5 lze vidět porovnání získaných hodnot ze senzoru lineárního zrychlení. Chůze simulovaná touto sadou parametrů má podobnou frekvenci dopadu nohou na zemi, což lze vidět pomocí počtu vrcholů signálu. Celková velikost vektoru zrychlení dosahuje nižších minimálních hodnot. Také lze vidět, že nejsilnější osou pohybu je X, zatímco v reálných datech je nejsilnější složkou osa Z. Toto je dáno výraznějším horizontálním pohybem a také rozdílnou rotací simulovaného a reálného zařízení. Rozdílná rotace je však očekávaná a chtěná vlastnost. Krom těchto rozdílů jsou si data průběhy a rozptylem hodnot podobná.

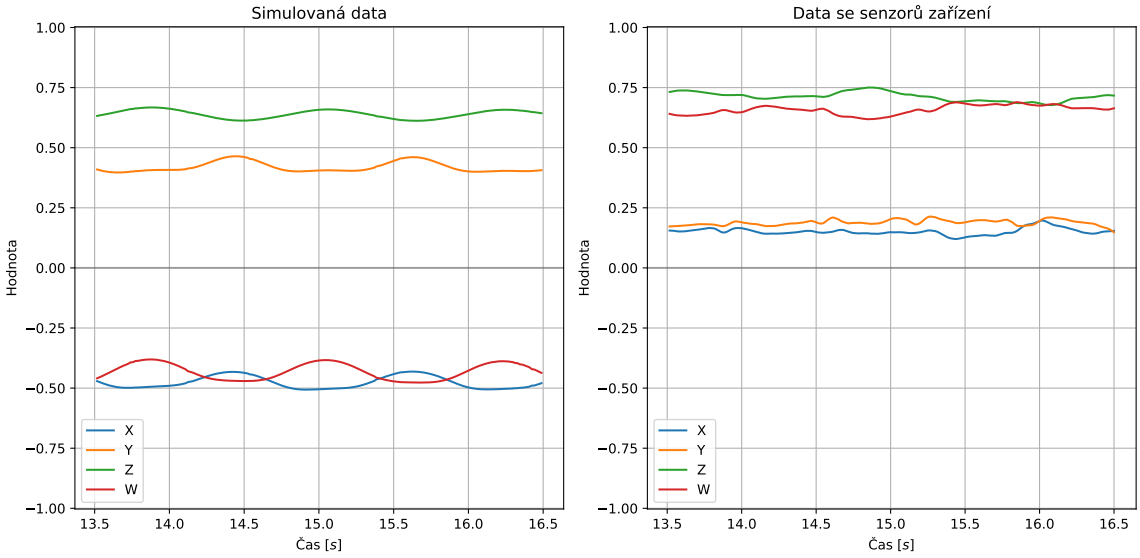
[2]Jméno vybrané sady parametrů chůze je *Walking-2024-10-5–01-02-39* a identifikátor porovnávaných reálných dat je *842c2cab-6179-4c91-bf03-75544f6f35a0*.



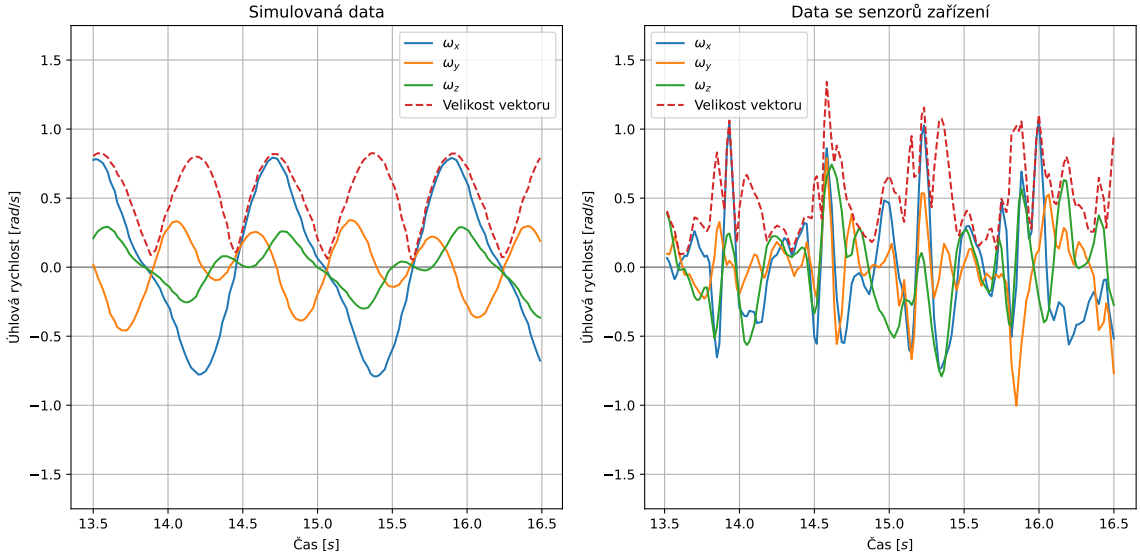
Obrázek 7.5: Porovnání senzoru hodnot lineárního zrychlení mezi reálnou a simulovanou chůzí

Na obrázku 7.6 lze vidět průběh rotace zařízení. Opět se liší absolutní hodnoty jednotlivých složek kvaternionu rotace, což je dané rozdílným simulovaným směrem zařízení. Důležité jsou jejich změny v čase. Oproti reálným datům se simulovaná rotace mění pozvolněji a je možné na ní pozorovat vliv sinusoid používaných pro generaci pohybu. Tato skutečnost je ještě více viditelná na obrázku 7.7. Oproti signálům reálného zařízení, které působí chaoticky a připomínají svými průběhy senzor zrychlení, jsou generované hodnoty gyroskopu velice pozvolné a je na nich velmi jednoduše vidět vliv jednotlivých sinusoid.

Z pozorovaných výsledků lze říci, že simulovaná poloha (a z ní počítané zrychlení) je velmi podobná reálným datům, ale rotace zařízení je až příliš pozvolná a pro přiblížení reálným datům by vyžadovala větší počet sinusoid, které na ni mají vliv.



Obrázek 7.6: Porovnání hodnot senzoru relativní orientace mezi reálnou a simulovanou chůzí



Obrázek 7.7: Porovnání hodnot gyroskopu mezi reálnou a simulovanou chůzí

Kapitola 8

Závěr

V této diplomové práci byla prozkoumána data získávaná z pohybových senzorů mobilního zařízení a možnosti simulace lidského pohybu. Na základě těchto průzkumů byl navržen způsob simulace lidského pohybu dopřednou kinematikou a tento návrh byl úspěšně implementován a integrován do webového rozšíření JShelter. Nakonec proběhlo experimentální otestování funkčnosti, výkonu a vhodnosti řešení.

Dopředná kinematika, která je používána pro pohyb kostry, je jedním ze standardních způsobů generace pohybu. Méně standardní je způsob získání vychýlení jednotlivých kostí skládáním sinusoid. Tento postup však umožňuje tvořit velké množství rozličných, spojitých a časem neomezených pohybů, mezi kterými je možné jednoduše a pozvolně přecházet.

V průběhu vývoje bylo očekáváno, že nejobtížnější částí implementace bude integrace do rozšíření JShelter. Tato integrace však proběhla téměř bez problémů, a naopak nečekaně obtížnou částí se stala tvorba objektivní funkce pro genetický algoritmus, který vytváří parametry pohybu. Z tohoto důvodu byly zatím vytvořeny objektivní funkce pouze pro 3 typy pohybu – chůze, sezení a zařízení položené na stole.

Výsledné řešení dokáže generovat pohyb velmi podobný pohybu zařízení v ruce člověka a hodnoty senzorů, které připomínají reálná data. V experimentech bylo zjištěno, že ve výsledcích simulace je možné pozorovat silný vliv některých sinusoid, které jsou ke generaci pohybu používány. Více realistických výsledků je možné dosáhnout např. generováním lepších parametrů, zvýšením počtu typů pohybu nebo zvýšením počtu kostí, ale i v současnosti je výsledné řešení velkým zlepšením oproti předchozí ochraně, která simulovala stacionární zařízení. Velkou výhodou je velmi malá časová náročnost výpočtu, která umožňuje použití této ochrany i na zařízeních s malým výkonem a zajišťuje zanedbatelný dopad na uživatelskou zkušenost při používání rozšíření JShelter.

Předávání falešných pohybových dat znemožňuje jejich zneužití k identifikaci uživatele napříč různými kontexty bez nutnosti zakázání použití senzorů, což by samo o sobě mohlo přispívat k tvorbě otisku. Pro ochranu soukromí uživatelů rozšíření JShelter jsou vytvořené změny přínosné, jelikož ztíží potenciálním hrozbám detekci ochrany senzorových dat. Pro detekci této ochrany by byla nutná analýza získávaných dat, oproti pouhé detekci pohybu. Tato oblast je však nekonečným závodem mezi obranou a útokem a každá vytvořená ochrana bude mít slabiny, nedokonalosti a rozpoznatelné vzory chování, které je možné detekovat a zneužít.

Literatura

[1] Albert, A. a Gerth, W. Analytic path planning algorithms for bipedal robots without a trunk. *Journal of Intelligent and Robotic Systems*. Springer. 2003, sv. 36, č. 2, s. 109–127.

[2] Aristidou, A., Lasenby, J., Chrysanthou, Y. a Shamir, A. Inverse kinematics techniques in computer graphics: A survey. In: Wiley Online Library. *Computer graphics forum*. 2018, sv. 37, č. 6, s. 35–58.

[3] Bayat, A. a Bayat, A. S. G. Classifying human walking patterns using accelerometer data from smartphone. *International Journal of Computer Science and Mobile Computing*. 2017, sv. 3, s. 1–6.

[4] Chung, S.-K. a Hahn, J. Animation of human walking in virtual environments. In: *Proceedings Computer Animation 1999*. 1999, s. 4–15. DOI: 10.1109/CA.1999.781194.

[5] Dekker, M. *Zero-moment point method for stable biped walking*. Technická zpráva 2009.072. Eindhoven University of Technology, 2009.

[6] GitHub. *About GitHub Pages* [online]. [cit. 2023-12-12]. Dostupné z: https://docs.github.com/en/pages/getting-started-with-github-pages/about-github-pages.

[7] Godot. *Introduction to Godot* [online]. 6. červenec 2023 [cit. 2023-12-16]. Dostupné z: https://docs.godotengine.org/en/stable/getting_started/introduction/introduction_to_godot.html.

[8] Han, J., Owusu, E., Nguyen, L. T., Perrig, A. a Zhang, J. ACComplice: Location inference using accelerometers on smartphones. In:. Leden 2012, s. 1–9. DOI: 10.1109/COMSNETS.2012.6151305.

[9] Kajita, S., Matsumoto, O. a Saigo, M. Real-time 3D walking pattern generation for a biped robot with telescopic legs. In: *Proceedings of the 2001 IEEE international conference on robotics and automation (Cat. no. 01ch37164)*. 2001, sv. 3, s. 2299–2306.

[10] Kajita, S. a Tani, K. Study of dynamic biped locomotion on rugged terrain-derivation and application of the linear inverted pendulum mode. In: *Proceedings of the 1991 IEEE International Conference on Robotics and Automation*. 1991, s. 1405–1406.

[11] Kolahi, A., Hoviattalab, M., Rezaeian, T., Alizadeh, M., Bostan, M. et al. Design of a marker-based human motion tracking system. *Biomedical Signal Processing and Control*. Elsevier. 2007, sv. 2, č. 1, s. 59–67.

[12] Kudoh, S. a Komura, T. C/sup 2/continuous gait-pattern generation for biped robots. In: *Proceedings of the 2003 IEEE/RSJ international conference on intelligent robots and systems (IROS 2003)(Cat. no. 03ch37453)*. 2003, sv. 2, s. 1135–1140.

[13] Lambora, A., Gupta, K. a Chopra, K. Genetic Algorithm- A Literature Review. In: *2019 International Conference on Machine Learning, Big Data, Cloud and Parallel Computing (COMITCon)*. 2019, s. 380–384. DOI: 10.1109/COMITCon.2019.8862255.

[14] Mehrnezhad, M., Toreini, E., Shahandashti, S. F. a Hao, F. Stealing PINs via mobile sensors: actual risk versus user perception. *International Journal of Information Security*. Springer. 2018, sv. 17, č. 3, s. 291–313.

[15] Michalevsky, Y., Boneh, D. a Nakibly, G. Gyrophone: Recognizing speech from gyroscope signals. In: *23rd USENIX Security Symposium (USENIX Security 14)*. 2014, s. 1053–1067.

[16] Mozilla. *Progressive web apps* [online]. říjen 2023 [cit. 2023-12-15]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/Progressive_web_apps.

[17] Mozilla. *Sensor APIs* [online]. Srpen 2023 [cit. 2023-11-26]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/API/Sensor_APIs.

[18] Multon, F., France, L., Cani Gascuel, M.-P. a Debunne, G. Computer animation of human walking: a survey. *The journal of visualization and computer animation*. Wiley Online Library. 1999, sv. 10, č. 1, s. 39–54.

[19] Mündermann, L., Corazza, S. a Andriacchi, T. P. The evolution of methods for the capture of human movement leading to markerless motion capture for biomechanical applications. *Journal of neuroengineering and rehabilitation*. BioMed Central. 2006, sv. 3, č. 1, s. 1–11.

[20] Park, J. H. a Kim, K. D. Biped robot walking using gravity-compensated inverted pendulum mode and computed torque control. In: *Proceedings of the 1998 IEEE International Conference on Robotics and Automation (Cat. No. 98CH36146)*. 1998, sv. 4, s. 3528–3533.

[21] Polčák, L., Saloň, M., Maone, G., Hranický, R. a McMahon, M. JShelter: Give Me My Browser Back. *Proceedings of the 20th International Conference on Security and Cryptography*. 2023, s. 287–294.

[22] Singh, G., Singla, A. a Virk, G. S. Modeling and simulation of a passive lower-body mechanism for rehabilitation. In: *Conference on mechanical engineering and technology (COMET-2016), IIT (BHU), Varanasi, India*. 2016.

[23] Sok, K. W., Kim, M. a Lee, J. Simulating biped behaviors from human motion data. In: *ACM SIGGRAPH 2007 papers*. 2007, s. 107–es.

[24] Spreitzer, R. PIN Skimming: Exploiting the Ambient-Light Sensor in Mobile Devices. In: *Proceedings of the 4th ACM Workshop on Security and Privacy in Smartphones & Mobile Devices*. New York, NY, USA: Association for Computing Machinery, 2014, s. 51–62. SPSM '14. DOI: 10.1145/2666620.2666622. ISBN 9781450331555. Dostupné z: https://doi.org/10.1145/2666620.2666622.

[25] Van Goethem, T., Scheepers, W., Preuveneers, D. a Joosen, W. Accelerometer-based device fingerprinting for multi-factor mobile authentication. In: Springer. *Engineering Secure Software and Systems: 8th International Symposium, ESSoS 2016, London, UK, April 6–8, 2016. Proceedings 8*. 2016, s. 106–121.

[26] Waldron, R. *Generic Sensor API* [online]. Candidate Recommendation. W3C, listopad 2023. Dostupné z: https://www.w3.org/TR/2023/CRD-generic-sensor-20231122/.

[27] Xiang, Y., Arora, J. S. a Abdel Malek, K. Physics-based modeling and simulation of human walking: a review of optimization-based and other approaches. *Structural and multidisciplinary optimization*. Springer. 2010, sv. 42, s. 1–23.

Příloha A

Obsah přiloženého paměťového média

- `/`
  - `readme.md` ... Textový soubor s návody ke spuštění a poznámkami
  - `thesis.pdf` ... Text diplomové práce
  - `bin` ... Složka obsahující sestavené verze programů
    - `jshelter_chrome.zip` ... Upravená verze rozšíření JShelter
    - `SensorDataSimulation` ... Aplikace pro generaci sad parametrů kostry
    - `SensorDataVisualisation` ... Aplikace pro zobrazení prostorových dat
  - `src` ... Složka obsahující zdrojové kódy
    - `jsrestrictor-sensors` ... Upravená verze rozšíření JShelter
    - `SensorDataCollecting` ... Webová aplikace pro sběr dat senzorů
    - `SensorDataSimulation` ... Aplikace pro generaci sad parametrů kostry
    - `SensorDataVisualisation` ... Aplikace pro zobrazení prostorových dat
    - `thesis` ... Zdrojový kód této diplomové práce
  - `data` ... Složka obsahující data této diplomové práce
    - `collectedData.json` ... Nasbíraná data ze senzorů
    - `accelerometerTestParams.json` ... Parametry pro test akcelerometru
    - `gyroscopeTestParams.json` ... Parametry pro test gyroskopu
    - `GeneratedParameters` ... Vygenerované sady parametrů pro pohyb kostry