UNIVERZA V LJUBLJANI

FAKULTETA ZA RAČUNALNIŠTVO IN INFORMATIKO

Miloš Vukov

Uporaba senzorja Leap Motion za brezdotično upravljanje spletnega brskalnika

DIPLOMSKO DELO

VISOKOŠOLSKI STROKOVNI ŠTUDIJSKI PROGRAM PRVE STOPNJE RAČUNALNIŠTVO IN INFORMATIKA

MENTOR: viš. pred. dr. Alenka Kavčič

Ljubljana 2014

Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Tržaška 25
1001 Ljubljana, Slovenija
telefon: 01 476 84 11
01 476 83 87
faks: 01 426 46 47
01 476 87 11
www.fri.uni-lj.si
e-mail: dekanat@fri.uni-lj.si

Št. naloge: 00528 / 2013
Datum: 14.9.2013

Univerza v Ljubljani, Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Kandidat: **MILOŠ VUKOV**

Naslov: **UPORABA SENZORJA LEAP MOTION ZA BREZDOTIČNO UPRAVLJANJE SPLETNEGA BRSKALNIKA**
USING LEAP MOTION FOR TOUCHLESS NAVIGATION IN A WEB BROWSER

Vrsta naloge: Diplomsko delo visokošolskega strokovnega študija prve stopnje

Tematika naloge:

Za interakcijo uporabnika z računalnikom je na voljo veliko načinov; klasičnemu, uporabi tipkovnice in miške, se v zadnjem času množično pridružujejo zasloni na dotik, pa tudi brezdotični načini, kot je na primer uporaba gest ali upravljanje z miselnimi ukazi. V okviru diplomske naloge proučite senzor Leap Motion, ki natančno zaznava položaj in premikanje uporabnikovih rok, in njegovo možno uporabo za upravljanje računalnika, predvsem za interakcijo s spletnimi brskalniki. Predstavite obstoječe pristope in možnosti njegove uporabe v povezavi s spletnimi brskalniki. Na podlagi ugotovitev razvijte razširitev za brskalnik Google Chrome, ki omogoča uporabo senzorja Leap Motion za sprehajanje po spletnih straneh. Implementirajte premikanje kazalca po zaslonu, klik na povezavo pod kazalcem ter prehode med odprtimi zavihki. Nalogo zaključite z oceno uporabnosti in robustnosti delovanja izdelane razširitve.

Mentor:
Alenka Kavčič
viš. pred. dr. Alenka Kavčič

Dekan:
prof. dr. Nikolaj Zimic

Fakulteta za računalništvo in informatiko • UNIVERZA V LJUBLJANI • 2

IZJAVA O AVTORSTVU DIPLOMSKEGA DELA

Spodaj podpisani Miloš Vukov, z vpisno številko **63990163**, sem avtor diplomskega dela z naslovom:

Uporaba senzorja Leap Motion za brezdotično upravljanje spletnega brskalnika

S svojim podpisom zagotavljam, da:

- sem diplomsko delo izdelal samostojno pod mentorstvom viš. pred. dr. Alenke Kavčič,
- so elektronska oblika diplomskega dela, naslov (slov., angl.), povzetek (slov., angl.) ter ključne besede (slov., angl.) identični s tiskano obliko diplomskega dela
- soglašam z javno objavo elektronske oblike diplomskega dela v zbirki "Dela FRI".

V Ljubljani, dne 13. marec 2014 Podpis avtorja:

KAZALO

KAZALO SLIK

SEZNAM UPORABLJENIH KRATIC

- **AJAX** (angl. *Asynchronous JavaScript and XML*) - Skupina tehnologij za ustvarjanje dinamičnih spletnih strani.
- **CSS** (angl. *Cascading Style Sheets*) - Kaskadne stilske predloge, ki določajo izgled spletnih strani.
- **CCD** (angl. *Charge Coupled Device*) - Optični senzor v digitalnih kamerah.
- **DLL** (angl. *Dynamic Link Libraries*) - Dinamično povezovalna knjižnica z izvršljivimi rutinami.
- **HTML** (angl. *Hyper Text Markup Language*) - Označevalni jezik za izdelavo spletnih strani.
- **HTML5** (angl. *Hyper Text Markup Language 5*) - HTML verzije 5.
- **JSON** (angl. *JavaScript Object Notation*) - Preprost format za izmenjavo podatkov.
- **SDK** (angl. *Software Development Kit*) - Paket orodij za razvoj programske opreme.
- **USB** (angl. *Universal Serial Bus*) - Univerzalno serijsko vodilo.
- **XML** (angl. *Extensible Markup Language*) - Razširljiv označevalni jezik.

POVZETEK

Cilj diplomske naloge je razvoj razširitve za spletni brskalnik Chrome, ki omogoča brezdotično upravljanje spletnega brskalnika s pomočjo senzorja Leap Motion. Predstavljene so zmogljivosti senzorja Leap Motion in njegovo delovanje. Analiza obstoječih programskih rešitev pokaže določene pomanjkljivosti, ki so upoštevane pri razvoju razširitve. Razširitev za brskalnik Chrome je razvita z orodjem Visual Studio in projektno predlogo za brskalnik Chrome. V izdelani razširitvi se uporabljajo dodatne programske knjižnice, ki vsebujejo funkcije, s katerimi se poenostavi razvoj rešitve. Predstavljeni so osnovni gradniki razširitve za brskalnik Chrome in arhitektura razvite razširitve. Opisane so implementirane funkcionalnosti razširitve in način izdelave. Izdelana razširitev omogoča brskanje po spletnih straneh z navigacijo po povezavah in premikom po zavihkih s pomočjo geste. Na koncu je podana ocena uporabnosti razširitve ter predlogi za nadaljnje delo.

Ključne besede: senzor, Leap Motion, brezdotični vmesnik, brskalnik, Chrome, razširitev, gesta.

ABSTRACT

The main objective of this diploma thesis is to develop a Chrome browser extension that would enable touchless navigation in web browser using the Leap Motion sensor. The paper thus assesses the capacity of the Leap Motion sensor and explains its functions. While the analysis of the existing software solutions shows certain shortcomings, they are, however, considered in the extension development process. The Chrome browser extension has been developed by using Visual Studio tools and the Chrome browser project template. The final, ready-made extension uses additional software libraries, containing functions to simplify the solution development. The paper introduces the basic components of the Chrome browser extension as well as the architecture model of the new one. What is more, it is devoted to a description of the functionalities implemented to improve the extension and guides us through the entire development process. The ready-made extension now enables internet browsing by navigating through the links and moving along the tabs with the help of gestures. Last but not least, the conclusion provides a utility assessment of the extension along with suggestions for further improvement.

Keywords: sensor, Leap Motion, touchless interface, web browser, Chrome, extension, gesture.

1 UVOD

V vsakdanjem življenju si ljudje medsebojno komunikacijo lajšajo in krajšajo s pomočjo gest. Za razliko od medsebojne komunikacije med ljudmi pri komunikaciji s strojno opremo večinoma uporabljamo ustaljene vmesnike, kot sta računalniška miška in tipkovnica, ki ne omogočata naravne interakcije. Vendar razvoj informacijske tehnologije in strojne opreme danes ustvarjata nove in boljše vmesnike, ki jih lahko uporabimo za interakcijo s strojno opremo. Z njihovo pomočjo lahko ustvarimo boljšo povezljivost med uporabnikom in strojno opremo ter delno ali popolnoma nadomestimo ustaljene vmesnike. Dva izmed takšnih vmesnikov sta senzorja Kinect in Leap Motion.

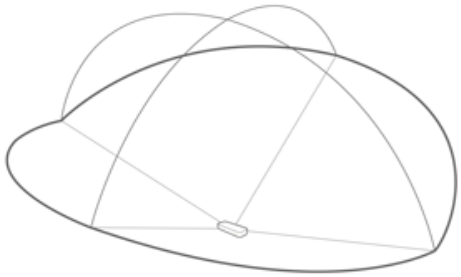
Senzor Kinect omogoča globinsko zaznavanje gibanja človeškega skeleta in prepoznavanje glasovnih ukazov v prostoru, ki je velikosti dnevne sobe, ter je bil ob izidu namenjen igranju računalniških iger. Kasneje so se pojavile rešitve, ki so omogočale interakcijo z drugimi vrstami programske opreme, kot je lahko na primer interakcija z oglasnim panojem. Za razliko od senzorja Kinect ima senzor Leap Motion manjše opazovano polje in natančneje zaznava položaj in premikanje uporabnikovih rok. Namenjen je za bližjo uporabo, torej neposredno pred zaslonom. Z njegovim natančnim zaznavanjem rok, prstov, orodij in gest lahko prepoznamo in izvajamo operacije, kot je na primer premikanje kazalca po zaslonu.

V okviru diplomske naloge smo izdelali razširitev za spletni brskalnik Chrome, ki omogoča brezdotično upravljanje brskalnika. V prvem delu je predstavljen senzor Leap Motion, njegov programski vmesnik in osnovni modeli, ki jih senzor podaja preko programskega vmesnika. V drugem delu je pregled obstoječih rešitev, kjer podamo kriterije za ocenjevanje. Temu sledi opis funkcionalnosti rešitev ter prednosti in slabosti za vsako rešitev posebej. V tretjem delu predstavimo razvojno okolje, programske knjižnice in osnovne gradnike razširitve za brskalnik Chrome. Sledi opis arhitekture in izvedbe razširitve za brskalnik Chrome, ki opisuje razvoj funkcionalnosti premikanja kazalca na zaslonu z njegovo kalibracijo, navigacije po povezavah ter premik po zavihkih. Zaključimo z oceno uporabnosti izdelane razširitve ter predlogi za nadaljnje delo.

2 SENZOR LEAP MOTION

Senzor Leap Motion je majhna USB naprava, ki se jo priključi na računalnik ter postavi pred zaslon, kot prikazuje slika 2.1, in omogoča uporabniku, da se z gestami rok sporazumeva s programsko opremo [1]. Senzor opazuje tridimenzionalni prostor nad napravo, ki je velik približno 240 cm^3 (prikazan na sliki 2.2), s pomočjo dveh CCD kamer in treh infrardečih senzorjev [2, 3]. Kameri zajemata dvodimenzionalno sliko, infrardeči senzorji pa informacije o globini, ki se nato procesirajo v servisu z imenom Leapd [4]. Ta opravi skoraj celotno procesiranje. Servis odstrani morebitne šume in zgradi model rok, prstov, gest in orodij, ki jih držimo [1]. Ti visokonivojski podatki so nam dostopni kot del okvirja (angl. *frame*), do njih pa lahko dostopamo s pomočjo programskega vmesnika in nam jih naprava podaja z maksimalno hitrostjo 290 okvirjev na sekundo. Senzor je zelo natančen, saj lahko sledi gibanju modelov z natančnostjo ene stotinke milimetra.

Slika 2.1: Senzor Leap Motion.



Slika 2.2: Opazovani prostor senzorja.

2.1 PROGRAMSKI VMESNIK

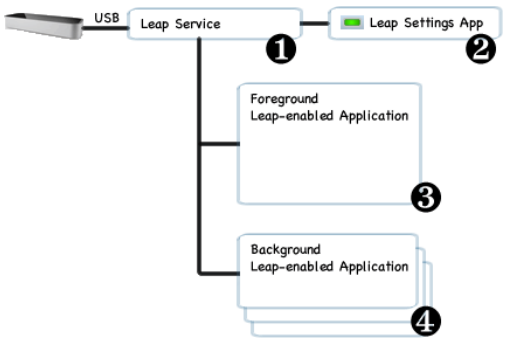
Razvijalci programske opreme se lahko povežejo s senzorjem ter pridobivajo podatke preko visokonivojskih programskih knjižnic, ki so del paketa razvojnih orodij (SDK). Programske knjižnice so na voljo za večino programskih jezikov (C++, C#, Objective-C, Java, Javascript, Python). Na podatkovni tok se lahko povežemo preko domorodnega programskega vmesnika (angl. *native application interface*) ali pa preko vmesnika WebSocket.

2.1.1 Domorodni programski vmesnik

Domorodni programski vmesnik je na voljo kot knjižnica DLL, ki se poveže na servis ter nudi podatke aplikaciji. Slika 2.3 prikazuje shemo povezav in je sestavljena iz naslednjih delov:

1. Servis pridobiva podatke iz senzorja preko vodila USB ter jih nato obdela in pošlje aplikacijam.
2. Aplikacija senzorja, kjer si lahko uporabnik nastavi uporabniške nastavitve za senzor.
3. Aktivna aplikacija, ki teče v ospredju ter komunicira s senzorjem. Naenkrat je lahko aktivna samo ena aplikacija [4].

4. Ko aplikacija izgubi fokus v operacijskem sistemu, ji servis preneha pošiljati podatke. Če želimo, da aplikacija, ki se izvaja v ozadju, prejema podatke iz servisa, je potrebno to omogočiti ob povezavi na servis [4].

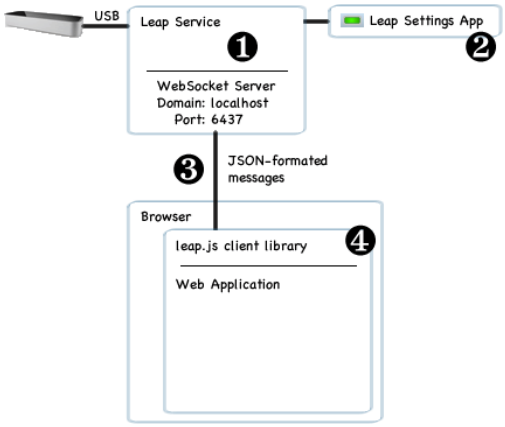


Slika 2.3: Shema povezav preko domorodnega programskega vmesnika.

2.1.2 Vmesnik WebSocket

Za spletne aplikacije je namenjen vmesnik WebSocket. Slika 2.4 prikazuje shemo povezav preko tega vmesnika in je sestavljena iz naslednjih delov:

1. Servis pridobiva podatke iz senzorja preko vodila USB ter jih nato obdela in ponuja preko strežnika WebSocket, ki sprejema povezave na lokalnem naslovu na vratih 6437.
2. Aplikacija senzorja, kjer si lahko uporabnik nastavi uporabniške nastavitve za senzor.
3. Server pošilja podatke v obliki JSON.
4. V brskalniku se z uporabo JavaScript knjižnice LeapJS povežemo na strežnik WebSocket ter prejemamo podatke. Programski vmesnik, ki ga ponuja knjižnica LeapJS, je podoben domorodnemu programskemu vmesniku.



Slika 2.4: Shema povezav preko vmesnika WebSocket.

Za povezavo na tok podatkov je potrebno ustvariti nadzornika (angl. *controller*), ki mu lahko podamo nastavitve. Nato se pripnemo na nadzornika in preko povratne funkcije (angl. *callback*) prejemamo podatke o okvirjih, kot kaže koda 2.1.

```
var controller = new Leap.Controller();
controller.on('animationFrame', (frame) =>
{
        this.Action(frame);
});
```

Koda 2.1: Povezava na senzor in prejemanje okvirjev.

Hitrost podajanja okvirjev lahko kontroliramo z uporabo argumenta frameEventName. Če uporabimo vrednost animationFrame, potem hitrost podajanja okvirjev upravlja brskalnik in je praviloma 60 okvirjev na sekundo. To je tudi priporočljiv način za prikaz animacij v brskalniku, saj se akcija izvede tik pred osvežitvijo vsebine v spletnem dokumentu. Če uporabimo vrednost deviceFrame, potem pridobivamo podatke z enako hitrostjo, kot jih podaja servis [5].

2.2 MODELI

Roka, kazalec in gesta so osnovni modeli z lastnostmi, ki jih podaja sistem Leap Motion preko programskih knjižnic.

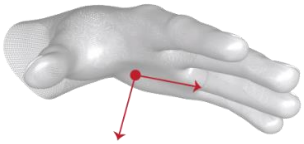
2.2.1 Roka

Model opazovane roke predstavlja fizično roko, ki jo je zaznal sistem Leap Motion, in vsebuje informacije o poziciji, orientaciji, karakteristikah in gibanju preko osnovnih lastnosti, ki so prikazane v tabeli 2.1 [6]. Sistem Leap Motion zazna največ dve roki istočasno [7] neglede na to, koliko jih je v opazovanem polju senzorja. Ob tem ne razlikuje med levo in desno roko. Do seznama rok (lastnost hands) dostopamo iz okvirja, ki ga prejemamo.

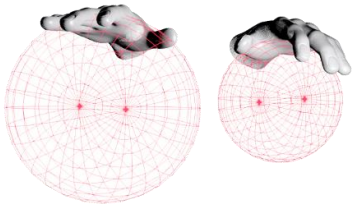
Lastnost	**Opis**
direction	Smerni vektor roke.
fingers	Seznam prstov.
palmNormal	Smerni vektor dlani.
palmPosition	Pozicija dlani.
palmVelocity	Hitrost premika roke.
pointables	Seznam kazalcev.
sphereCenter	Pozicija krogle.
sphereRadius	Polmer krogle.
timeVisible	Količina časa, ko je bila roka nepretrgoma vidna.
tools	Seznam orodij.

Tabela 2.1: Seznam osnovnih lastnosti modela roke.

Orientacijo roke lahko določimo s pomočjo dveh vektorjev, ki ju vidimo na sliki 2.5. Smerni vektor roke (lastnost direction) podaja smer, v katero je usmerjena roka. Smerni vektor dlani (lastnost palmNormal) pa podaja smer, v katero je obrnjena dlan. Na sliki 2.6 vidimo navidezno kroglo, ki jo določata lastnosti sphereCenter in sphereRadius, s katerima lahko na primer ugotovimo, ali je uporabnik želel izvesti prijem.



Slika 2.5: Smerna vektorja dlani in roke.



Slika 2.6: Zmanjševanje velikosti virtualne krogle ob navideznem prijemanju.

2.2.2 Kazalec

Preko modela roke dostopamo do kazalcev, ki pripadajo roki in so v vidnem polju senzorja. Nekaj osnovnih lastnosti kazalca je podanih v tabeli 2.2 [8]. Kazalci so lahko prsti ali pa orodja. Sistem Leap Motion razlikuje med orodjem in prstom po tem, da je orodje tanjše, daljše in bolj ravno kot prst, kar je prikazano na sliki 2.7.

Lastnost	**Opis**
direction	Smerni vektor kazalca.
length	Dolžina kazalca.
stabilizedTipPosition	Stabilizirana lokacija kazalca.
tipPosition	Pozicija kazalca.
tipVelocity	Hitrost premika kazalca.
tool	Ali je kazalec orodje.

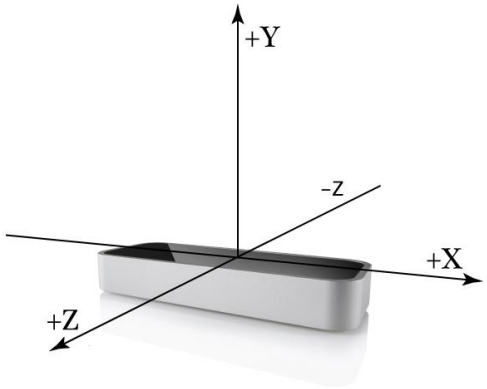
Tabela 2.2: Seznam osnovnih lastnosti modela kazalec.

Pozicija kazalca (lastnost tipPosition) je točka na koncu orodja ali prsta. Točka prsta je prikazana na sliki 2.8 s smerjo (lastnost direction), v katero kaže.

Slika 2.7: Orodje je tanjše, daljše in bolj ravno kot prst.

Slika 2.8: Pozicija prstov ter smerni vektorji prstov.

Sistem Leap Motion podaja pozicijo kot točko v tridimenzionalnem kartezičnem koordinatnem sistemu, ki ima središče postavljeno na sredino senzorja z negativno osjo Z, ki je usmerjena proti zaslonu [9], kot je prikazano na sliki 2.9.



Slika 2.9: Koordinatni sistem.

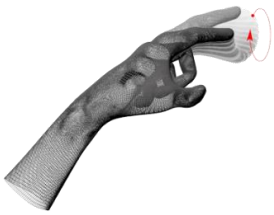
2.2.3 Gesta

Senzor opazuje aktivnosti v opazovanem polju ter poizkuša prepoznati vzorec, ki predstavlja gesto [10]. Ko sistem Leap Motion prepozna vgrajeno gesto, jo doda na seznam gest (lastnost gestures) v okvirju. Osnovne lastnosti modela gest so prikazane v tabeli 2.3.

Lastnost	**Opis**
duration	Količina časa izvajana geste.
handsIds	Seznam identifikatorjev rok.
pointableIds	Seznam identifikatorjev kazalcev.
state	Stanje geste.
type	Tip geste.

Tabela 2.3:Seznam osnovnih lastnosti modela gesta.

Trenutno ima sistem vgrajeno prepoznavo gest kroženja, zamaha, klika na tipko in klika na zaslon. Gesta kroženje, ki predstavlja kroženje kazalca (kot je razvidno na sliki 2.10), in gesta zamah, ki predstavlja ravni zamah roke z iztegnjenimi prsti (kot je razvidno na sliki 2.11), sta dlje trajajoči gesti, ki imata lahko stanja začetek (angl. *start*), posodobitev (angl. *update*) in konec (angl. *stop*). S pomočjo stanja geste se lahko odzovemo na primer z začetkom animacije, ko se je gesta pričela, ali pa izvedemo akcijo geste, ko se je ta končala izvajati.

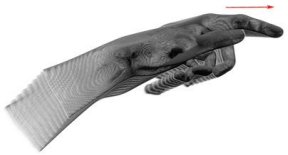


Slika 2.10: Gesta kroženje s kazalcem.

Slika 2.11: Gesta horizontalni zamah.

Preostali dve gesti, klik na tipko in klik na zaslon, ki ju prikazujeta slika 2.12 in slika 2.13, sta diskretni gesti, ki imata samo eno stanje. Na diskretno gesto se lahko odzovemo, ko se je ta izvedla, torej ima stanje vrednost stop.

Slika 2.12: Gesta klik na tipko.



Slika 2.13: Gesta klik na zaslon.

Za zaznavanje gest je pri inicializaciji nadzornika potrebno podati parameter enableGestures z vrednostjo true [11]. Potem pridobimo poslušalca z metodo gesture, ki ji podamo tip geste. Za zaznavo geste klik na tipko, kot smo že omenili zgoraj, se je potrebno odzvati ob koncu izvedbe geste s pomočjo metode stop, ki ji podamo povratno funkcijo za izvedbo akcije, kot prikazuje koda 2.2.

```
var controller = new Leap.Controller({ enableGestures: true });

var screenTapListener = controller.gesture('keyTap');
screenTapListener.stop((gesture) =>
{
	this.Action(gesture);
});
```

Koda 2.2: Detekcija geste klik na tipko.

Sistem Leap Motion ne omogoča enostavne definicije geste po meri, pri kateri bi se prepoznava izvajala v servisu ter podajala v seznamu gest. Prepoznavanje geste (enoročne ali dvoročne) po meri je potrebno izvajati v programu z razpoložljivimi podatki, ki jih vsebuje okvir. Pri tem je potrebno v spletnih rešitvah biti zelo pazljiv in optimizirati prepoznavo, saj lahko v nasprotnem primeru izvajanje JavaScript kode obremeni brskalnik do te mere, da rešitev ni uporabna.

3 PREGLED OBSTOJEČIH REŠITEV

Pregledali in analizirali smo uporabniško prijaznost rešitev, ki so trenutno na voljo v okolju Windows za brezdotično brskanje po spletnih straneh na osebnem računalniku z uporabo senzorja Leap Motion. Pri tem smo si zastavili naslednje kriterije:

- Enostavna začetna uporaba rešitve brez dodatnega nastavljanja.
- Čim bolj enostavne in naravne geste, ki namigujejo na tip akcije.
- Vizualizacija začetka izvajanja oz. izvedbe akcije.
- Zadovoljivo število akcij za neprekinjeno brezdotično brskanje po spletnih straneh.
- Natančno in tekoče izvajanje.

Po analizi smo ugotovili, da nobena rešitev ne izpolnjuje vseh kriterijev. Kot največja napaka izstopa neusklajenost kazalca na zaslonu s prstom. Pri testiranju smo zelo neintuitivno premikali kazalec na zaslonu. Ko smo želeli izvesti kalibracijo, je nekatere rešitve sploh niso imele implementirane. Tam, kjer je funkcionalnost kalibracije bila prisotna, pa ni delovala oz. ni ponudila pričakovanega rezultata.

3.1 DEXTYPE FOR LEAPMOTION

Rešitev je razvita kot razširitev za brskalnik Chrome ter je na voljo za namestitev preko spletne trgovine Chrome [12]. Je brezplačna in omogoča naslednje funkcionalnosti:

- Pisanje besedila.
- Premikanje kazalca.
- Izvajanje navigacije.
- Zapiranje zavihkov.
- Odpiranje novih zavihkov.
- Premikanje po zavihkih.
- Osveževanje strani.

Rešitev začnemo uporabljati tako, da najprej pokažemo s prstoma obeh rok proti zaslonu. Nato se vklopi razširitev in prikaže akcijski menu v zgornjem desnem kotu znotraj okna, kar prikazuje slika 3.1. V oknu se pojavita dva kazalca v obliki kroga, s katerima izvajamo akcije, kot je na primer klik na povezavo. Akcije se izvajajo po metodi pokaži in suni (angl. *point and poke*). Z desnim kazalcem pokažemo na povezavo ter z drugim sunemo proti zaslonu. Ostale akcije izvajamo preko menuja.

Slika 3.1: Ukazni menu razširitve DexType for LeapMotion.

Rešitev omogoča tudi prostoročno pisanje besedila s pomočjo virtualne tipkovnice na dnu okna, pri čemer poizkuša samodejno prepoznati besedo, ki smo jo hoteli napisati. Rešitev poskuša prepoznavati besedo tudi v primeru, ko smo se zmotili. Podpira 31 jezikov, med njimi tudi slovenščino [13].

Dobre strani rešitve so:

- Tekoče premikanje kazalcev.
- Funkcionalnost pisanja besedila.
- Ni očitnega postopka kalibracije.
- Natančen klik na povezavo s sistemom pokaži in suni.

Slabe strani rešitve so:

- Neusklajenost kazalca na zaslonu s prstom.
- Ostale akcije se izvajajo preko menuja namesto z namenskimi gestami.

3.2 LEAPTOUCH V.1

Rešitev [14] je razvita kot razširitev za brskalnik Chrome ter je na voljo brezplačno za namestitev preko spletne trgovine Chrome in omogoča naslednje funkcionalnosti:

- Prikaz pomoči.
- Izvajanje navigacije.
- Kalibracija kazalca.
- Zapiranje zavihka.
- Odpiranje novih zavihkov.
- Premikanje po zavihkih.
- Vertikalni pomik po vsebini.
- Horizontalni pomik po vsebini.
- Zapiranje brskalnika.

Pri tej rešitvi so akcije vezane na gesto. Za prikaz pomoči tako razpremo vse prste in prikaže se pomoč znotraj zavihka, ki je prikazana na sliki 3.2. Za obisk povezave pokažemo s kazalcem na povezavo ter pomaknemo prst proti zaslonu.

Slika 3.2: Pomoč, ki se prikaže znotraj zavihka, v programu LeapTouch.

Rešitev vsebuje tudi čarovnika za kalibracijo. Ob zagonu čarovnika se v oknu pojavijo tri točke, v katere posamično usmerimo prst. Na koncu kalibracije lahko izvedemo tudi fino kalibracijo s smernimi tipkami, s katerimi poravnamo prst in kazalec na zaslonu.

Dobre strani rešitve so:

- Klik na povezavo z gesto.
- Intuitivne geste.
- Enostaven in hiter prikaz pomoči.

Slabe strani rešitve so:

- Kalibracija deluje slabo, pri premikanju se pojavi zelo očitno neujemanje kazalca na zaslonu in prsta.
- Izvedba klika na navigacijski element je težavna, saj potrebuješ zelo mirno roko za izvedbo.
- Moteče zaostajanje kazalca pri hitrejših premikih.

3.3 LEAPUI

Rešitev je razvita kot razširitev za brskalnik Chrome in kot samostojna knjižnica za JavaScript [15, 16]. Knjižnica se imenuje LeapuiJS in je namenjena integraciji spletnih strani s senzorjem Leap Motion z enostavno vključitvijo knjižnice v spletno stran. Obe rešitvi sta na voljo brezplačno in omogočata naslednje funkcionalnosti:

- Premikanje kazalca znotraj zavihka.
- Vertikalni pomik po vsebini.
- Pomik na začetek vsebine.

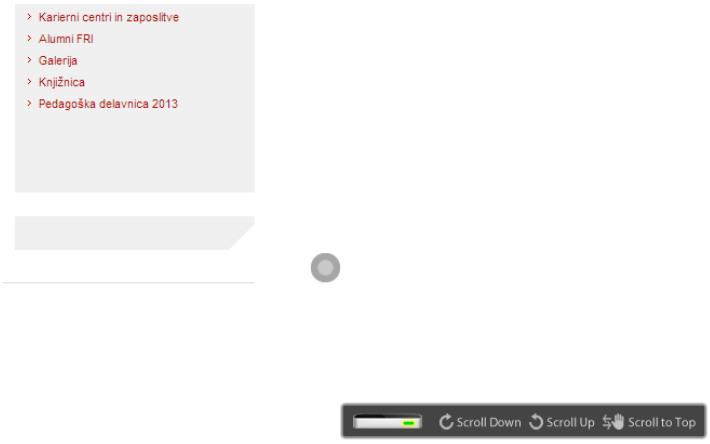
Rešitev je zelo enostavna in ima zelo osnovne funkcionalnosti. Trenutno lahko z omenjenimi funkcionalnostmi brskamo znotraj enega zavihka, saj ne omogoča premikanja po zavihkih. O zagonu rešitve se v spodnjem desnem kotu zavihka pojavi informativna pasica, ki podaja informacije o možnih gestah, kot prikazuje slika 3.3.

Dobre strani rešitve so:

- Kazalec deluje zelo tekoče.
- Neujemanje prsta in kazalca je sprejemljivo, glede na to da ni kalibracije.

Slabe strani rešitve so:

- Zelo malo funkcionalnosti.
- Vertikalni pomik po vsebini ne deluje najbolje.



Slika 3.3: Informativna pasica v programu LeapUI.

3.4 OSControl – Chrome Edition

Rešitev je razvita kot samostojna aplikacija za okolje Windows 7 (ali novejše) ter je na voljo za namestitev preko spletne trgovine AirSpace [17]. Rešitev ne poizkuša nadomestiti računalniške miške in tipkovnice, ampak je zasnovana tako, da je njun naravni podaljšek. Je plačljiva ter omogoča naslednje funkcionalnosti:

- Premikanje kazalca po celotnem zaslonu.
- Odpiranje novih zavihkov v brskalniku Chrome.
- Tiskanje/shranjevanje strani.
- Preklop brskalnika v celozaslonski način.

Aplikacija se nahaja v območju za obvestila (angl. *notification area*) ter čaka na gesto. Lahko tudi popolnoma nadomesti računalniško miško, če želimo samo premikati kazalec na zaslonu in se pomikati po izbrani vsebini.

Dobre lastnosti rešitve so:

- Miškin kazalec je na voljo za celoten sistem in ne samo za brskalnik.
- Zaradi implementacije rešitve izven brskalnika lahko sama aplikacija dostopa do programskega vmesnika Windows in pridobi natančnejše informacije o sistemu (na primer informacijo o velikosti zaslona).

Slabe lastnosti rešitve so:

- Ni prenosljiv na druge operacijske sisteme.

4 RAZVOJ RAZŠIRITVE ZA SPLETNI BRSKALNIK

Pri analizi obstoječih rešitev smo bili najbolj nezadovoljni z uporabniško izkušnjo kazalca, klika na povezavo in kalibracijo. Poizkušali bomo izboljšati oz. podati informacije o tem, kakšne težave se pojavijo ter kako jih rešiti, če želimo izkušnjo izboljšati. Ob načrtovanju rešitve smo se odločili, da izdelamo rešitev, ki jo bo mogoče uporabiti na različnih operacijskih sistemih. Tako smo se odločili, da izdelamo razširitev za brskalnik Chrome (angl. *Chrome extension*).

4.1 RAZVOJNO OKOLJE

Razvoj razširitve je potekal na platformi Windows s pomočjo razvojnega orodja Visual Studio in uporabo različnih spletnih tehnologij, knjižnic in predlog, ki so v nadaljevanju na kratko predstavljeni.

4.1.1 Visual Studio

Razvojno orodje Microsoft Visual Studio 2013 (v nadaljevanju Visual Studio) omogoča hiter razvoj rešitev z različnimi pomagali, kot so barvanje, sintaktično preverjanje in pametno dopolnjevanje (angl. *intellisense*) kode, ter predlogami za različne vrste projektov, ki so na voljo preko razširitev (angl. *extensions*) [18].

4.1.2 Projektna predloga za razširitev brskalnika Chrome

Ker smo razvijali razširitev za Chrome, smo si namestili tudi projektno predlogo Google Chrome Extension Project Template, s katero dobimo že pripravljeno osnovno strukturo imenikov, osnovnih datotek in zmožnost pakiranja razširitve za objavo na spletni trgovini Chrome (angl. *Chrome web store*) [19].

4.1.3 TypeScript

Namesto dinamičnega jezika JavaScript (v nadaljevanju JavaScript) smo izbrali TypeScript, ki prinaša strogo tipiziranost (angl. *static typing*) in razredno usmerjenost objektov (angl. *class-based object orientation*) z namenom ustvarjanja nadgradljivih visoko zmogljivih (angl. *scalable high-performance*) programov [20]. TypeScript sintaksa je nadmnožica standarda ECMAScript 5 z nekaterimi predlaganimi funkcionalnostmi standarda ECMAScript 6. Sintaktične izboljšave s seboj prinašajo poznane pojme, kot so razred (angl. *class*), modul in vmesnik (angl. *interface*), katerih JavaScript trenutno ne pozna. TypeScript prevajalnik se lokalno izvede nad datoteko s končnico ts in poizkuša ne spreminjati vrstnega reda kode in imen spremenljivk zato, da je prevedena koda kar se da ujemajoča in enostavna za razhroščevanje [21]. V razvojnem okolju Visual Studio se koda TypeScripta, ki je prikazana v kodi 4.1, ob shranjevanju prevede v JavaScript. Prevedeno kodo nato uporabimo v spletnih rešitvah in je prikazana v kodi 4.2. Generirajo se tudi datoteke s končnico js.map, ki povezujejo JavaScript kodo s TypeScript kodo za potrebe razhroščevanja v brskalnikih.

```
class Greeter
{
	greeting: string;
	constructor(message: string)
	{
		this.greeting = message;
	}
	greet()
	{
		return "Hello, " + this.greeting;
	}
}
var greeter = new Greeter("world");
var msg = greeter.greet();
```

Koda 4.1: Koda, napisana v jeziku TypeScript.

```
var Greeter = (function ()
{
	function Greeter(message)
	{
		this.greeting = message;
	}
	Greeter.prototype.greet = function ()
	{
		return "Hello, " + this.greeting;
	};
	return Greeter;
})();
var greeter = new Greeter("world");
var msg = greeter.greet();
```

Koda 4.2: JavaScript koda, ki je prevedena iz TypeScript kode.

4.1.4 UnderscoreJS

UnderscoreJS je JavaScript knjižnica, ki ponuja splošne funkcionalnosti za hitrejši razvoj v JavaScriptu z zbirkami, objekti in funkcijami [22]. V okviru naloge smo uporabili metodi throttle in debounce.

4.1.5 AmplifyJS

AmplifyJS je zbirka komponent, ki rešuje splošne težave pri razvoju spletnih aplikacij z zelo preprostim programskim vmesnikom. V okviru naloge smo uporabili funkcionalnost sporočilnega vzorca objavi in naroči (angl. *publish and subscribe*), ki je mnogo lažja za uporabo kot implementacija v JavaScript knjižnici JQuery [23].

4.1.6 JQuery

JQuery je majhna in hitra knjižnica za skriptni jezik JavaScript in vsebuje že vnaprej spisane funkcije, s katerimi zelo pospešimo razvoj programske opreme. V sklopu funkcionalnosti omogoča sprehajanje po elementih, rokovanje z dogodki, animacije in preprosto uporabo tehnologije AJAX, ki delujejo ne glede na brskalnik, v katerem se izvaja [24].

4.2 RAZŠIRITEV ZA BRSKALNIK CHROME

Razširitev za brskalnik Chrome je skupek datotek HTML, CSS, JavaScript in ostalih vrst datotek, ki jih potrebujemo. Ker je to v osnovi spletni dokument, lahko izrablja tehnologije, kot so AJAX, JSON in HTML5 [25].

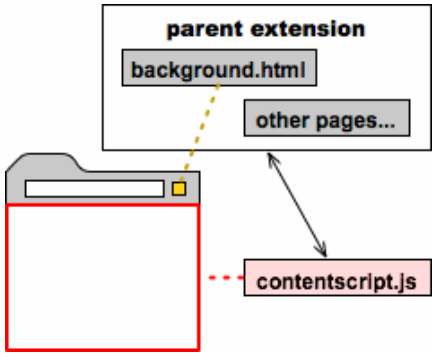
Dodatek sestavljajo:

- Nevidna stran, ki se izvaja v ozadju (angl. *background page*).
- Skripta vsebine (angl. *contents script*).
- Strani, ki predstavljajo uporabniški vmesnik razširitve.

Poznamo dve vrsti strani v ozadju: vztrajne (angl. *persistent background pages*) in dogodkovne (angl. *event pages*). Kot že ime pove, se vztrajne strani naložijo, ko se naloži razširitev. Dogodkovne strani se naložijo, ko je to potrebno. Nevidna stran se izvaja v kontekstu razširitve in lahko dostopa do celotnega programskega vmesnika.

Skripta vsebine je JavaScript skripta, ki se izvaja v kontekstu spletne strani in dostopa do njenih informacij ter jo lahko spreminja. Dostopa lahko samo do programskega vmesnika chrome.extension, ki je zelo omejen ter ponuja metode za komunikacijo z nevidno stranjo in dostop do virov (kot so na primer slike) razširitve. Direktna komunikacija med skriptami ni mogoča, lahko pa komunicirajo med seboj preko nevidne strani z izmenjavo sporočil.

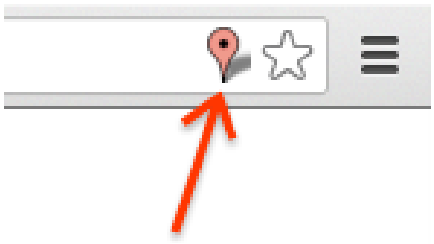
Na sliki 4.1 so prikazani gradniki razširitve in povezave med njimi: nevidna stran (background.html), skripta vsebine (contentsscript.js), spletna stran (rdeč okvir) in akcija brskalnika (rumen kvadratek).



Slika 4.1: Shema povezav med gradniki razširitve.

V brskalniku lahko prikažemo uporabniški vmesnik razširitve preko akcije brskalnika ali akcije strani. Za akcijo brskalnika se odločimo, ko je funkcionalnost primerna za večino strani in je na voljo na desni strani naslovne vrstice (kar prikazuje slika 4.2). Akcija brskalnika lahko na primer prikaže pojavno okno (angl. *popup*), ki prikaže HTML dokument. Za akcijo strani se odločimo takrat, ko je potrebno, da se akcija pojavi oz. izgine odvisno od spletne strani. Primer akcije strani je viden na sliki 4.3. Vsaka razširitev ima lahko stran za nastavitve, ki je ravno tako HTML dokument, preko katerega nastavimo razširitev. HTML strani znotraj razširitve imajo polni dostop do objektnega modela spletnega dokumenta in ga lahko modificirajo.

Slika 4.2: Akcija brskalnika, ki prikaže pojavno okno.



Slika 4.3: Akcija strani, ki se lahko pojavi za določeno spletno stran.

Poleg dostopa do programskega vmesnika, ki ga ponujajo spletne strani in aplikacije, lahko razširitev uporablja tudi programski vmesnik brskalnika Chrome, ki omogoča boljšo integracijo z brskalnikom. Večina metod programskega vmesnika je asinhrona, kar pomeni, da se dejansko izvajanje operacije izvede v ozadju. Če želimo izvedeti, kdaj se konča, je potrebno podati povratno funkcijo, ki se izvede ob koncu asinhrone operacije in lahko ponudi dodatne informacije o izvedeni operaciji. Enostavni primer takega klica je prikazan v kodi 4.3.

```
chrome.tabs.query({ windowId: chrome.windows.WINDOW_ID_CURRENT }, (tabs) =>
{
    console.log("Number of tabs:" + tabs.length);
});
```

Koda 4.3: Primer klica asinhrone metode za pridobivanje zavihkov v trenutnem oknu.

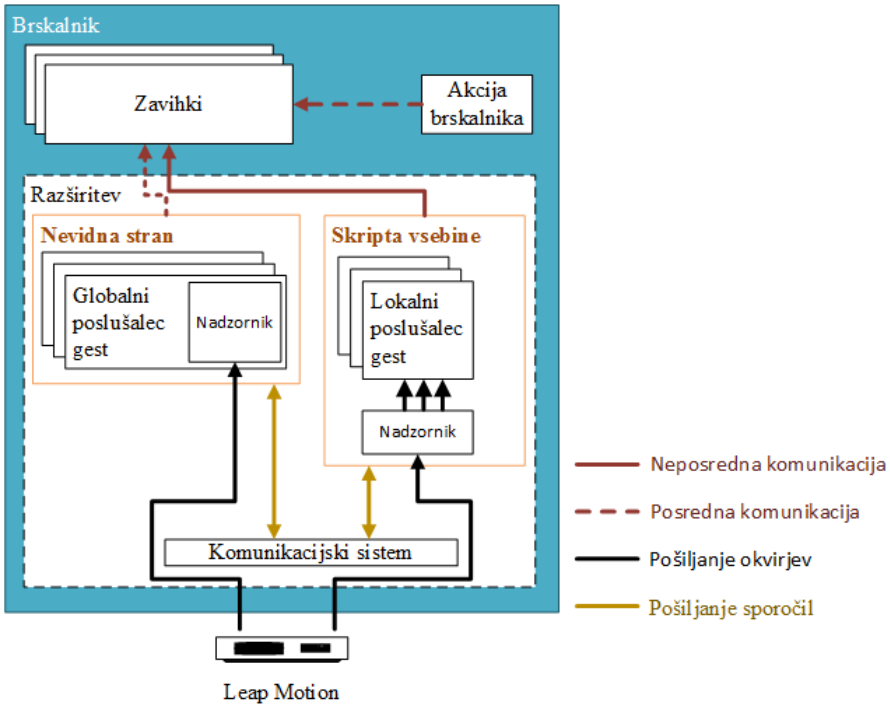
4.3 ARHITEKTURA

Nizkonivojska arhitektura predstavlja gradnike razširitve v povezavi z zavihki in senzorjem Leap Motion ter komunikacijo med njimi, kot je to prikazano na sliki 4.4.

Zagon rešitve poteka preko nevidne strani, ki se izvaja v ozadju. Ustvari se komunikacijsko vozlišče, naložijo se globalni poslušalci (angl. *listeners*) gest in nastavi se akcijo brskalnika, s pomočjo katere lahko poženemo čarovnika za kalibracijo. Vsi poslušalci gest morajo implementirati vmesnik IInitializer, ki vsebuje metodo Initialize, preko katere se zaženejo poslušalci gest. Vsak izmed globalnih poslušalcev ustvari svojega nadzornika s parametrom, da želi prejemati podatke, čeprav deluje v ozadju.

Komunikacijsko vozlišče je del komunikacijskega sistema, ki implementira sporočilni vzorec objavi-naroči (angl. *publish-subscribe*), s katerim dosežemo šibko sklopljenost med entitetami, kot so na primer poslušalci gest. Entiteta objavi sporočilo določenega tipa in ga po komunikacijskem vozlišču razpošlje entitetam, ki so se naročile na ta tip sporočila. Razpošiljanje sporočila je lahko lokalno znotraj skripte vsebine ali pa globalno med vsemi skriptami vsebine in nevidno stranjo. Komunikacijski sistem tako znotraj izrablja JavaScript knjižnico AmplifyJS za lokalno izmenjavo sporočil in programski vmesnik brskalnika Chrome za globalno izmenjavo sporočil.

Ob odprtju spletne strani v zavihku se ustvari instanca skripte vsebine, ki teče v kontekstu spletne strani. Ob tem se ustvari povezava na komunikacijsko vozlišče, ustvari se skupnega nadzornika in naloži se lokalne poslušalce gest. Vsi poslušalci prejmejo preko metode Intialize skupnega nadzornika, ki prejema podatke samo takrat, ko je zavihek aktiven.



Slika 4.4: Nizkonivojska arhitektura s prikazom komunikacije med gradniki rešitve.

4.4 FUNKCIONALNOSTI

V naši razširitvi smo implementirali naslednje funkcionalnosti: premik kazalca, kalibracija, navigacija po povezavah in premik po zavihkih, ki so opisane v nadaljevanju.

4.4.1 Premik kazalca

Premikanje kazalca je osnova za kasnejše obiskovanje povezav. Pri tem smo želeli, da se kazalec premika gladko in kar se da natančno.

Implementacijo navigacije sestavlja:

- Prikaz in pomikanje kazalca na zaslonu.
- Pretvorba točke, ki jo podaja senzor Leap Motion, v točko v zavihku.
- Detekcija prsta, s katerim kažemo na zaslon.

Najprej smo poizkušali risati kazalec na elementu HTML canvas, ki je namenjen risanju s pomočjo JavaScripta. Vendar je bilo pri uporabi te metode preveč težav, saj je potrebno skrbeti za čiščenje oz. osveževanje risalne površine. To lahko storimo tako, da izbrišemo element canvas ter ga ustvarimo vsakič znova ali pa da prerišemo staro grafiko. Ob izbrisu in ustvarjanju

novega elementa je prihajalo do neželenega učinka utripanja. Ob prerisu stare grafike pa do večje obremenitve procesorja.

Zato smo uporabili način, pri katerem vrinemo element (v našem primeru je to HTML element div, ki ima za ozadje sliko kazalca) v spletno stran ter ga s pomočjo pozicijskih CSS lastnosti premikamo po oknu. Kazalec je implementiran v razredu Pointer. Ta razred skrbi za premikanje in omogoča spremembo tipa kazalca. Implementirana sta dva tipa kazalca, to je osnovi kazalec in kazalec za označevanje povezave. Primer osnovnega kazalca, s katerim se pomikamo po spletni strani, prikazuje slika 4.5.

Slika 4.5: Osnovni kazalec, s katerim se pomikamo po spletni strani.

Za pravilen prikaz kazalca na zaslonu je potrebno preslikati pozicijo prsta na kazalec na zaslonu. Pozicijske enote, ki jih podaja senzor, so v milimetrih. Zato je potrebno najprej pretvoriti milimetre v piksle. Razmerje je mogoče pridobiti na več načinov. Najenostavnejši je ta, da s pomočjo CSS lastnosti narišemo črto velikosti 100 mm ter iz lastnosti offsetWidth pridobimo njeno dolžino v pikslih, kar kasneje delimo z dolžino črte. Enostavni primer pridobivanja razmerja prikazuje koda 4.4.

```
<html>
<body>
    <div id="line"
         style="margin: 0px; border: 0px; padding: 0px;
                border-top: 1px solid green; width: 100mm;" />
    <script>
        var px = document.getElementById("line").offsetWidth;
        var pxPerMm = px / 100;
    </script>
</body>
</html>
```

Koda 4.4:Enostavni primer pridobivanja razmerja (število pikslov na mm).

Nato je potrebno prenesti to točko na zaslon. Pri tem predpostavimo, da je uporabnik postavil senzor Leap Motion pod zaslon na sredino.

Glede na zgornjo predpostavko izračunamo x koordinato točke v brskalniku (x_d) tako, kot prikazuje enačba 4.1. s_w je širina celotnega zaslona v pikslih, w_{sl} je razdalja med levim robom zaslona in levim robom okna brskalnika v pikslih in x_{lm} je x koordinata pozicije prsta, kot nam jo podaja senzor Leap Motion po pretvorbi v piksle.

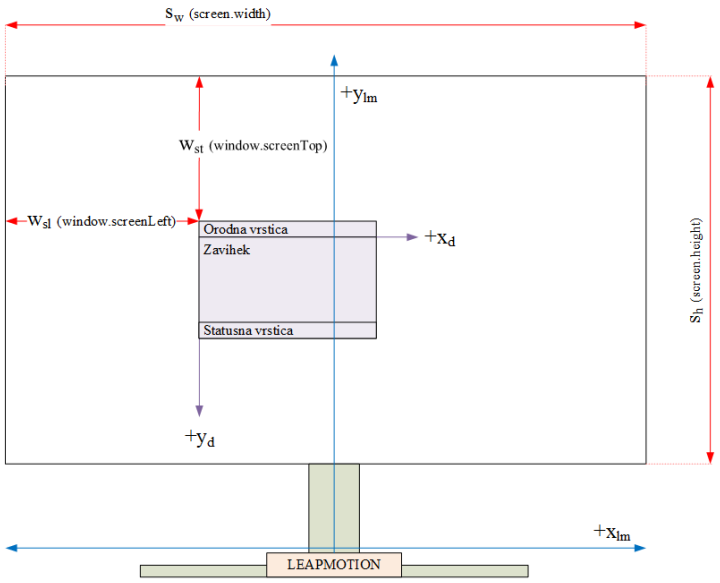
$$x_d = \frac{s_w}{2} - w_{sl} + x_{lm}$$

Enačba 4.1: Izračun koordinate x kazalca v oknu brskalnika.

Y koordinato točke v brskalniku (y_d) izračunamo, kot prikazuje enačba 4.2. Pri tej enačbi predpostavimo, da je senzor tik pod robom zaslona, višino orodne in statusne vrstice okna brskalnika pa zanemarimo. s_h je višina zaslona, w_{st} je razdalja med zgornjim robom zaslona in zgornjim robom brskalnika in y_{lm} je y koordinata pozicije prsta, kot nam jo podaja senzor Leap Motion po pretvorbi v piksle. Za boljšo predstavo so parametri za izračun točke v brskalniku prikazani na sliki 4.6 (na sliki 4.6 je senzor Leap Motion postavljen pod zaslon).

$$y_d = s_h - w_{st} - y_{lm}$$

Enačba 4.2: Izračun koordinate y kazalca v oknu brskalnika.



Slika 4.6: Grafični prikaz parametrov, potrebnih za izračun točke v brskalniku.

Za premikanje kazalca je potrebno poslušati podatke o premiku prsta. Ker pri tem spreminjamo pozicijo kazalca in mora brskalnik to spremembo osvežiti na zaslonu, ustvarimo nadzornika s parametrom animationFrame.

Pri testiranju premika kazalca se je pojavil neželeni učinek repa kazalca zaradi visoke frekvence osveževanja brskalnika (60 okvirjev na sekundo ali več), ob katerem se izvaja koda za premik kazalca, in počasnega izvajana JavaScript kode. Zato smo znižali frekvenco izvajanja premika kazalca s pomočjo funkcije throttle na 20 okvirjev na sekundo.

Pri premikanju kazalca se je pojavil tudi problem preskakovanja kazalca na zaslonu, saj smo na začetku predpostavljali, da če pokažemo samo z enim prstom na zaslon, nam bo naprava podala v seznamu prstov samo enega z indeksom 0. Vendar naprava zazna tudi druge skrčene

prste in napake, ki so lahko na prvem mestu v seznamu. Tako je bilo potrebno izračunati, kateri prst je najbližje zaslonu, in pozicijo tega prsta vzeti za kazalec.

4.4.2 Kalibracija

Takšen izračun točke kazalca v brskalniku, kot smo ga uporabili zgoraj, je dokaj natančen za horizontalno os. Pri vertikalni osi pride do precejšnega zamika zaradi orodne in statusne vrstice brskalnika ter višino med senzorjem in spodnjim delom zaslona (predpostavili smo, da je senzor tik pod zaslonom), ki jih v prejšnjih enačbah nismo upoštevali. Zaradi implementacije rešitve kot razširitve za brskalnik do informacij o višini orodne in statusne vrstice ne moremo dostopati. Tako je potrebno implementirati funkcionalnost poravnave kazalca s prstom (v nadaljevanju fina kalibracija).

Funkcionalnost je implementirana v razredu FineCalibration. Fino kalibracijo izvedemo tako, da pokažemo s prstom proti zaslonu ter s pomočjo smernih tipk (A – levo, S – dol, D – desno, W – gor) premikamo kazalec na zaslonu ter ga poravnamo s prstom. Pri tem razred FineCalibration preko sporočilnega sistema sporoča skripti vsebine razliko po horizontalni in vertikalni osi. Na koncu se vrednosti shranijo v lokalno shrambo podatkov in obvesti se ostale instance skript vsebine o novih vrednostih. Te vrednosti se nato upoštevajo pri izračunu točke brskalnika, kot prikazujeta enačba 4.3 in enačba 4.4.

$$x_d = \frac{s_w}{2} - w_{sl} + x_{lm} + x_{diff}$$

Enačba 4.3: Izračun koordinate x v oknu brskalnika z upoštevanjem fine kalibracije.

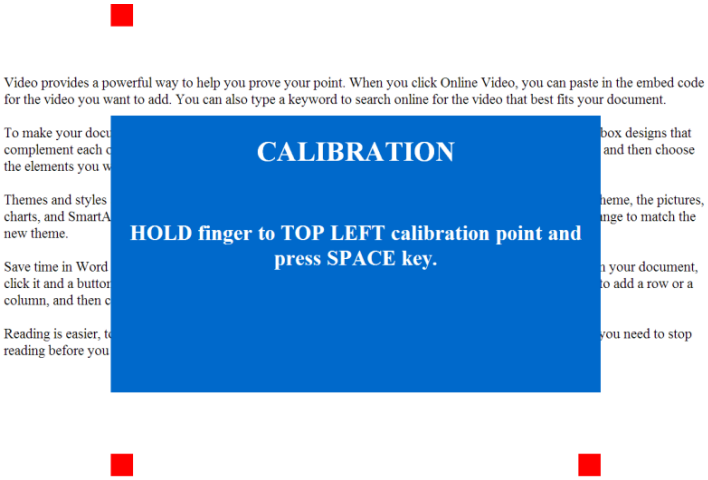
$$y_d = s_h - w_{st} - y_{lm} + y_{diff}$$

Enačba 4.4: Izračun koordinate y v oknu brskalnika z upoštevanjem fine kalibracije.

Kljub fini kalibraciji opažamo neujemanje kazalca s prstom, ko je ta na skrajnem robu brskalnika. Do največjega odstopanja pride po horizontalni osi, če smo fino kalibracijo izvedli na enem robu zaslona, nato pa se pomaknemo s prstom na nasprotni rob zaslona. Težava je v metodi, s katero smo izračunali razmerje. Absolutna CSS enota milimeter, ki smo jo uporabili za izris črte, ni namenjena za prikaz na zaslonu, ampak za prikaz na tiskanem mediju [26], kjer je gostota pik znana. Brskalnik Chrome pred izrisom na zaslon pretvori milimetre v piksle z gostoto 96 dpi (angl. *dot per inch*) [27] in jo prikaže. Ker ima večina zaslonov večjo gostoto pik, se nam prikaže krajša črta na zaslonu (na tiskanem mediju bi bila prave velikosti). Zaradi tega kazalec na zaslonu zaostaja. Zato smo implementirali še začetno kalibracijo, pri kateri dobimo dokaj natančno razmerje med milimetri in piksli.

Ko uporabnik postavi senzor Leap Motion pred zaslon, ta ni poravnan (pozitivna os y koordinatnega sistema Leap Motion ni pravokotna na površino zaslona) z zaslonom. Pri prenosnih računalnikih je lahko zaslon nagnjen nazaj in sta razmerji po vertikalni in horizontalni osi različni.

Za začetek kalibracije je potrebno klikniti na akcijo brskalnika, ki sproži čarovnika in se v aktivnem zavihku prikažejo tri kalibracijske točke, kot je to vidno na sliki 4.7.



Slika 4.7: Čarovnik za kalibracijo s tremi kalibracijskimi točkami.

Potrebno je pokazati s prstom na vsako izmed kalibracijskih točk in jih potrditi s pritiskom na tipko za presledek.

Za izračun razmerja po horizontalni osi je potrebno izračunati razdaljo med x koordinato spodnje leve točke in x koordinato spodnje desne točke v koordinatnem sistemu senzorja Leap Motion in koordinatnem sistemu brskalnika. S pridobljenimi razdaljami nato izračunamo, koliko pikslov je potrebnih za en milimeter po horizontalni osi. Enačba 4.5 prikazuje izračun razmerja za horizontalno os. $cp2_x$ je x vrednost spodnje leve kalibracijske točke, $cp3_x$ je x vrednost spodnje desne kalibracijske točke, $f2_x$ je x vrednost pozicije prsta kalibracijske točke spodaj levo in $f3_x$ je x vrednost pozicije prsta kalibracijske točke spodaj desno.

$$x_r = \frac{|cp2_x - cp3_x|}{|f2_x - f3_x|}$$

Enačba 4.5: Izračun razmerja za horizontalno os.

Za izračun razmerja po vertikalni osi je potrebno izračunati razdaljo med y koordinato zgornje leve točke in y koordinato spodnje leve točke v koordinatnem sistemu senzorja Leap Motion in koordinatnem sistemu brskalnika. S pridobljenimi razdaljami nato izračunamo, koliko pikslov je potrebnih za en milimeter po vertikalni osi. Enačba 4.6 prikazuje izračun razmerja za vertikalno os. $cp2_y$ je y vrednost spodnje leve kalibracijske točke, $cp1_y$ je y vrednost zgornje leve kalibracijske točke, $f1_y$ je y vrednost pozicije prsta kalibracijske točke zgoraj levo in $f2_y$ je y vrednost pozicije prsta kalibracijske točke spodaj levo.

$$y_r = \frac{|cp1_y - cp2_y|}{|f1_y - f2_y|}$$

Enačba 4.6: Izračun razmerja za vertikalno os.

Razmerji se nato shranita v lokalno shrambo podatkov ter osvežita v vseh instancah skripte vsebine.

4.4.3 Navigacija po povezavah

Za navigacijo po povezavah se uporabi vgrajeno gesto klik na zaslon.

Implementacijo navigacije sestavlja:

- Detekcija geste klik na zaslon.
- Detekcija, ali je nad elementom, na katerega kaže kazalec, mogoče izvesti klik.
- Vizualizacija, kadar je nad elementom mogoče izvesti klik.
- Izvedba akcije.

Za detekcijo geste ponuja programski vmesnik posebno metodo gesture, kateri podamo tip geste, ki jo želimo ujeti. Na vgrajeno gesto klik na zaslon se odzovemo tako, da podamo povratno funkcijo metodi stop, ki naj izvede akcijo navigacije (kar prikazuje koda 4.5).

```
var screenTapListener = controller.gesture('screenTap');
screenTapListener.stop((gesture) =>
{
    this.Action(gesture);
});
```

Koda 4.5: Primer kode, ki izvede akcijo ob gesti klik na zaslon.

Za detekcijo, ali je možno nad pokazanim HTML elementom izvesti klik, preverimo, ali velja vsaj eden izmed pogojev:

- Element ima definiran atribut href.
- Element ima definiran atribut onclick.
- Element ima definiran stil CSS z vrednostjo pointer:cursor.

Ko se zazna navigacijski element, se uporabniku zamenja tip kazalca, ki nakazuje, da lahko izvede akcijo, kot kaže slika 4.8. Za uporabniško izkušnjo je namreč zelo pomembno, da se vizualizira možnost navigacije nad elementom. Za detekcijo navigacijskega elementa skrbi razred MovePointer, ki preko sporočilnega podsistema obvesti razred Navigate o možnosti izvedbe navigacije.

Slika 4.8: Kazalec za označevanje povezave.

Problem pri detekciji geste je ta, da se lahko gesta v zelo kratkem času sproži večkrat. Večkrat se proži zaradi tega, ker sistem Leap Motion lahko eno uporabnikovo gesto zazna kot

več zaporednih gest. Posledično se lahko pri izvedbi geste klik na zaslon večkrat izvede obisk povezave. To težavo odpravimo s funkcijo debounce, ki zaduši večkratno izvajanje akcije.

Za samo izvedbo navigacije nad elementom uporabimo metodo click, ki je na voljo v JavaScript knjižnici JQuery in ki pravilno sproži dogodek click nad izbranim elementom. Ob tem se pri počasnejših izvedbah navigacije prikaže tudi animacijo klika, s čimer uporabnika obvestimo o izvajanju.

4.4.4 Premik po zavihkih

Funkcionalnost premikanja po zavihkih je sestavljena iz:

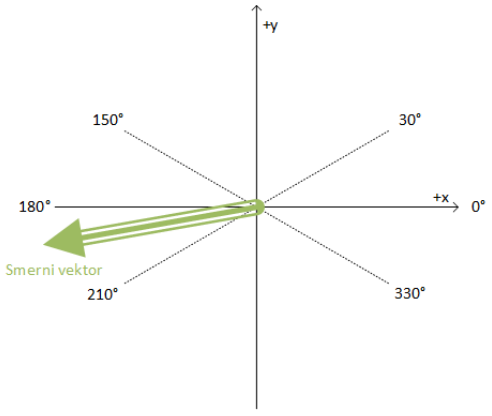
- Detekcije geste zamah v levo stran.
- Premika na prejšnji zavihek.

Za premik po zavihkih brskalnika smo uporabili vgrajeno gesto zamah. Detekcija vgrajenih gest je hitrejša in manj obremenjujoča za proces brskalnika, saj se zaznavanje geste izvaja v servisu. Za detekcijo, kdaj se gesta izvede, je potrebno ustvariti poslušalca s parametrom swipe. Nato pa podati povratno funkcijo metodi stop, ki izvede akcijo. Koda 4.6 prikazuje skrajšani primer uporabe.

```
var swipeListener = controller.gesture('swipe');
swipeListener.stop((gesture) =>
{
        this.Action(gesture);
});
```

Koda 4.6: Koda za akcijo, ko uporabnik izvede celotni zamah.

Ker se gesta sproži za vse smeri, se preveri, da je bila gesta izvedena od desne proti levi z dlanjo, obrnjeno v smeri zamaha (zamah z desno roko v levo stran). Smerni vektor geste pridobimo iz lastnosti direction. Smerni vektor dlani pa iz lastnosti palmNormal. Oba smerna vektorja pravokotno projiciramo na ravnino xy ter izračunamo njun kot v tej ravnini. Z idealnim (ravnim) zamahom v levo stran bi imela oba smerna vektorja kot 180 stopinj. Vendar idealnega zamaha nikoli ne izvedemo, zato preverimo, če sta oba pridobljena kota med 150 in 210 stopinj (slika 4.9 prikazuje ravnino xy s smernim vektorjem, ki kaže v levo stran). Območje med 150 in 210 stopinj smo določili eksperimentalno (zamahi desne roke v levo stran).



Slika 4.9: Smerni vektor v ravnini xy.

Tudi pri tej zaznavi geste imamo enak problem, kot je pri navigaciji po povezavah: sproži se lahko večkrat in to iz dveh razlogov. Prvi razlog je, da lahko sistem Leap Motion interpretira eno gesto kot več zaporednih gest. Drugi razlog pa je, da je gesta zamah vezana na zamah prsta in nam ob zamahu z roko sproži geste za vsak prst posebej. Za odpravo neželenega učinka je potrebno obravnavati množico kretenj, ki se zgodijo v zelo kratkem času, kot eno samo gesto. To težavo odpravimo z uporabo funkcije debounce, ki počaka določen čas po zadnji gesti in šele nato izvede akcijo skoka na prejšnji zavihek ter s tem zaduši izvajanje kasnejših gest.

Po zaznavi geste se aktivira prejšnji zavihek. S pomočjo programskega vmesnika brskalnika Chrome pridobimo seznam zavihkov v trenutnem oknu in poiščemo prejšnjega ter nastavimo, da je aktiven. Aktiviranje prejšnjega zavihka s pomočjo programskega vmesnika Chrome prikazuje koda 4.7, pri čemer je currentIndex indeks zavihka, ki je trenutno aktiven.

```
chrome.tabs.update(tabs[currentIndex - 1].id, { active: true });
```

Koda 4.7: Aktiviranje prejšnjega zavihka.

Implementiran je samo premik na prejšnji zavihek. Na podoben način se naredi premik na naslednji zavihek. Pri zamahu leve roke v desno stran se preveri, da sta kota smernih vektorjev med 0 in 30 stopinj ali med 330 in 360 stopinj. Za aktiviranje naslednjega zavihka se trenutnemu indeksu prišteje vrednost ena.

5 SKLEP

V okviru diplomske naloge smo izdelali razširitev za brskalnik Chrome, ki omogoča brezdotično brskanje po spletnih straneh. Razširitev omogoča osnovne operacije za brskanje po spletnih straneh, kot je premikanje kazalca, obiskovanje povezav ter premikanje na prejšnji zavihek. Z izdelavo razširitve smo želeli izboljšati pomanjkljivosti, ki smo jih opazili ob pregledu obstoječih rešitev.

Na začetku razvoja smo morali raziskati implementacijo osnovnih funkcionalnosti in poiskati ustrezne rešitve (kot je na primer premikanje kazalca po zaslonu). Končna rešitev je dokaj robustna in dobro deluje v različnih pogojih in za različne uporabnike, tako da bi jo lahko uporabili tudi za brezdotični kiosk. Kazalec deluje tekoče in dokaj natančno. Brez velikega truda deluje klikanje po večjih povezavah, ki v neposredni bližini nimajo drugih povezav. Za klikanje po manjših povezavah ali po povezavah, ki se držijo skupaj, je še vedno potrebna večja natančnost pri gesti.

Zastavljena implementacija ponuja še veliko možnosti za nadgradnjo in izboljšave. Predvsem zaradi aktivnega razvoja knjižnice LeapJS in posodobitev njene dokumentacije.

5.1 PREDLOGI ZA NADALJNJE DELO

Izdelano razširitev bi lahko nadgradili in izboljšali na več področjih. V nadaljevanju je navedenih nekaj predlogov za nadaljnje delo.

Pri povečavi strani se poveča tudi naš kazalec. Pri izrisu kazalca bi bilo potrebno upoštevati faktor povečave ter ustrezno pomanjšati naš kazalec. Informacijo o faktorju povečave lahko pridobimo iz objektnega modela dokumenta. Potrebno bi bilo tudi dopolniti kodo za prikaz kazalca v razredu Pointer, saj je velikost kazalca trenutno fiksna.

Knjižnica LeapJS se je v zadnjem času posodobila in sedaj omogoča tudi drugačen način preslikave točk na zaslon s pomočjo razreda InteractionBox. Potrebno bi bilo pregledati možnost uporabe omenjenega razreda v ta namen.

Posodobljena dokumentacija je v zadnjem času prinesla tudi razdelek o posnemanju dotika (angl. *touch emulation*). Lahko bi preverili, ali je razred TouchZone, ki je v njej omenjen, boljša izbira za izvajanje klika na zaslon, saj dokumentacija opisuje tak primer uporabe.

Kot smo že omenili, je klikanje manjših ali tistih povezav, ki se držijo zelo skupaj, oteženo, saj mora biti uporabnik bolj natančen pri izvajanju geste klik na zaslon. V takih primerih se nam lahko zgodi, da kliknemo na sosednjo povezavo oz. moramo gesto večkrat izvesti, ker ob izvedbi pozicija našega prsta ni bila nad povezavo. Klikanje po manjših povezavah bi lahko izboljšali tako, da bi dodali geste, s katerimi bi manipulirali povečanje oz. pomanjšanje vsebine strani. S povečanjem strani bi se povečala tudi povezava in oddaljenost od sosednjih povezav, s čimer bi se tega problema znebili. Prav tako bi lahko povečali (poudarili, da izstopa) le tisto povezavo, nad katero je trenutno kazalec. Vendar bi morali kazalec zaradi povečane povezave bolj premakniti, da bi prišli do druge povezave.

Spletne strani ob povezavah ponujajo uporabniku še druge možnosti za interakcijo. To so okvirčki z namigom (angl. *tooltip*) in spustni menuji, ki se aktivirajo, ko gre uporabnik preko njih z računalniško miško. Potrebno bi bilo ugotoviti, kako lahko prepoznamo take vrste elementov in kako jih kar najbolj intuitivno aktiviramo z gesto.

Razširitev bi lahko nadgradili še tako, da bi omogočala premikanje na naslednji zavihek. Nadgradnja bi bila enostavna, saj bi bilo potrebno samo dopolniti preverjanje, ali se je zamah zgodil v desno stran, in se pri tem odzvati s premikom na naslednji zavihek.

6 LITERATURA

[1] M. Spiegelmock, *Leap Motion Development Essentials*, Birmingham : Packt Publishing, 2013.

[2] (2014) Buying a Leap Motion Controller. Dostopno na: http://support.leapmotion.com/entries/29601858-Buying-a-Leap-Motion-Controller

[3] (2014) The unocial Leap FAQ. Dostopno na: https://forums.leapmotion.com/forum/general-discussion/general-discussion-forum/434-the-unocial-leap-faq?420-The-unocial-Leap-FAQ=

[4] (2014) Websocket protocol for communicating with leapd. Dostopno na: https://github.com/leapmotion/leapjs/blob/master/PROTOCOL.md

[5] (2014) LeapJS - Getting Frames. Dostopno na: https://developer.leapmotion.com/leapjs/tutorials/getting-frames

[6] (2014) Hand - Leap Motion Documentation. Dostopno na: https://developer.leapmotion.com/documentation/javascript/api/Leap.Hand.html

[7] (2014) Leap Motion - Developer Guide - Hand. Dostopno na: https://developer.leapmotion.com/leapjs/api-guide#Leap_Hand

[8] (2014) Pointable - Leap Motion Documentation. Dostopno na: https://developer.leapmotion.com/documentation/javascript/api/Leap.Pointable.html

[9] (2014) API Overview - Leap Motion documentation. Dostopno na: https://developer.leapmotion.com/documentation/Languages/JavaScript/Guides/Leap_Overview.html

[10] (2014) Gesture Leap Motion Documentation. Dostopno na: https://developer.leapmotion.com/documentation/javascript/api/Leap.Gesture.html

[11] (2014) Leap Motion - Tutorial - Gestures. Dostopno na: https://developer.leapmotion.com/leapjs/tutorials/gestures

[12] (2014) DexType for LEAP Motion. Dostopno na: https://chrome.google.com/webstore/detail/dextype-for-leap-motion/nmdipjbemendfkmjpebpokbbmghdligc

[13] (2014) DexType FAQ. Dostopno na: http://dextype.com/faq/

[14] (2014) Leap Touch. Dostopno na: https://chrome.google.com/webstore/detail/leap-touch/fomagommnhckeikpfbeddjojfpdmhcmh

[15] (2014) Leap UI. Dostopno na: https://chrome.google.com/webstore/detail/leap-ui/bfjdefghccpahlcnalhlbmaioeelganf

[16] (2014) LEAP UI - the Easiest integration of web and Motion Control. Dostopno na: http://www.leapui.net/

[17] (2014) Leap Motion Airspace App Store - OSControl - Chrome Edition. Dostopno na: https://airspace.leapmotion.com/apps/oscontrol-chrome-edition/windows

[18] B. Johnson, *Professional Visual Studio 2012*, Indianapolis: John Wiley & Sons, 2012.

[19] (2014) Google Chrome Extension Project Template. Dostopno na: http://visualstudiogallery.msdn.microsoft.com/c11cd639-2abb-4243-96d2-153c0adb494a

[20] (2014) Welcome to TypeScript. Dostopno na: http://www.typescriptlang.org/

[21] (2014) S. Fenton, SyncFucion Ebooks TypeScript Succinctly. Dostopno na: http://www.syncfusion.com/resources/techportal/ebooks/typescript

[22] (2014) UnderscoreJS. Dostopno na: http://underscorejs.org/

[23] (2014) Pub/Sub API AmplifyJS. Dostopno na: http://amplifyjs.com/api/pubsub/

[24] J. Chaffer in K. Swedberg, Learning jQuery, Third Edition, Birmingham : Packt Publishing, 2011.

[25] (2014) Overview - Google Chrome. Dostopno na: http://developer.chrome.com/extensions/overview

[26] (2014) CSS: EM, PX, PT, CM, IN. Dostopno na: http://www.w3.org/Style/Examples/007/units.en.html

[27] (2014) CSS Values and Units Module Level 3. Dostopno na: http://www.w3.org/TR/css3-values/#absolute-length-units