Unos teksta zasnovan na praćenju pomaka oka: prediktivno modeliranje i empirijsko istraživanje

Katalinić, David

SVEUČILIŠTE U RIJECI
TEHNIČKI FAKULTET
Sveučilišni diplomski studij računarstva

Diplomski rad

Unos teksta zasnovan na praćenju pomaka oka: prediktivno modeliranje i empirijsko istraživanje

Rijeka, rujan 2024.

David Katalinić
0069087868

SVEUČILIŠTE U RIJECI
TEHNIČKI FAKULTET
Sveučilišni diplomski studij računarstva

Diplomski rad

Unos teksta zasnovan na praćenju pomaka oka: prediktivno modeliranje i empirijsko istraživanje

Mentor: izv.prof.dr.sc. Sandi Ljubić

Rijeka, rujan 2024.

David Katalinić
0069087868

SVEUČILIŠTE U RIJECI
TEHNIČKI FAKULTET
POVJERENSTVO ZA DIPLOMSKE ISPITE

Rijeka, 06.03.2024.

Zavod: Zavod za računarstvo

Predmet: Interakcija čovjeka i računala

ZADATAK ZA DIPLOMSKI RAD

Pristupnik: **David Katalinić (0069087868)**

Studij: Sveučilišni diplomski studij računarstva (1400)

Modul: Računalni sustavi (1442)

Zadatak: **Unos teksta zasnovan na praćenju pomaka oka: prediktivno modeliranje i empirijsko istraživanje / Eye-Tracking-based Text Entry: Predictive Modeling and Empirical Research**

Opis zadatka:

U kontekstu virtualnih on-screen tipkovnica za sustave u desktop okruženju, osmisliti minimalno dva rješenja/modaliteta za unos teksta koja se zasnivaju na konceptu praćenja pomaka očiju. S obzirom na poznatu geometriju predloženih tipkovnica, frekvenciju pojavljivanja digrama u ciljanom jeziku, te eksperimentalno utvrđene vremenske značajke operacije pokazivanja-i-odabira u dotičnom kontekstu, izračunati predviđene gornje granice brzine unosa teksta. Organizirati i provesti HCI eksperiment sa stvarnim korisnicima, s ciljem uvida u vrijednosti metrika WPM i TER za sve kombinacije razina nezavisnih varijabli. Usporediti rezultate prediktivnog modeliranja i empirijskog istraživanja.

Rad mora biti napisan prema Uputama za pisanja diplomskih / završnih radova koje su objavljene na mrežnim stranicama studija.

Zadatak uručen pristupniku: 20.03.2024.

Predsjednik povjerenstva za
diplomski ispit:
prof. dr. sc. Miroslav Joler

Mentor:
izv. prof. dr. sc. Sandi Ljubić

Izjava o samostalnoj izradi rada

Izjavljujem da sam samostalno izradio ovaj rad.

Rijeka, rujan 2024.

_____

David Katalinić

Sadržaj

Popis slika

Poglavlje 1

Uvod

Glavna tema koja se obrađuje u ovom radu je unos teksta zasnovan na praćenju pomaka oka. Razvijena su dva modaliteta interakcije: slobodna putanja pokazivača i pomak lijevo-desno uz dvije vrste virtualnih tipkovnica: tipkovnica s kvadrantima i QWERTY tipkovnica. Provedeno je eksperimentalno vrednovanje i prediktivno modeliranje s ciljem uvida u brzinu unosa teksta pomoću metrike WPM (engl. *Words Per Minute, WPM*) i ukupna stopa pogreške (engl. *total error rate, TER*).

Prije govora o samoj implementaciji potrebno je predstaviti sam unos teksta zasnovan na praćenju pogleda koji je obrađen u sljedećem poglavlju. Nakon toga u trećem poglavlju opisan je razvoj i značajke metoda za unos teksta kao i korišteni tehnološki stog aplikacije. U četvrtom poglavlju detaljno je opisan postupak provedbe prediktivnog i eksperimentalnog vrednovanja kao i korištenih aplikacija. Dodatno, opisan je Fittsov zakon i njegova uloga u prediktivnom modeliranju. U petom poglavlju prikazani su rezultati za WPM, TER i percipirano radno opterećenje interakcije. Također, opisani su i statistički testovi koji su korišteni u analizi podataka. Na kraju petog poglavlja prikazani su rezultati prediktivnog modeliranja te je dana usporedba rezultata empirijskog istraživanja i prediktivnog modeliranja.

Poglavlje 2

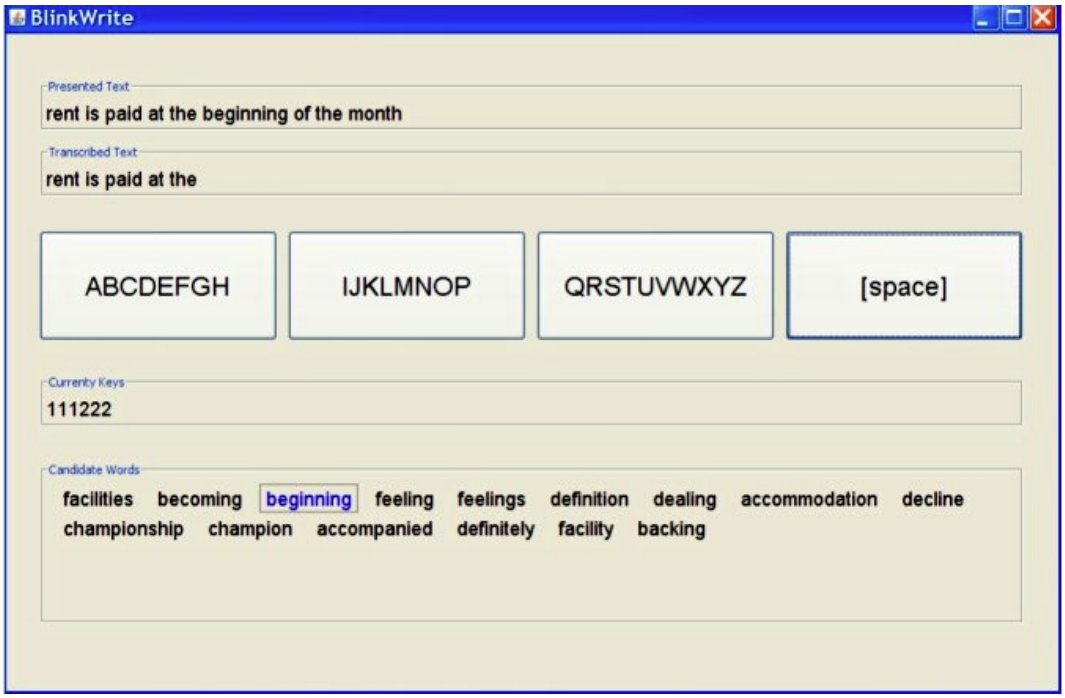
Unos teksta zasnovan na praćenju pomaka oka

Unos teksta zasnovan na praćenju pomaka oka je oblik asistivne tehnologije koji je ključan za osobe s ograničenim motoričkim sposobnostima. Korištenje ove tehnologije omogućuje osobama koje pate od ALS-a ili sindroma zaključanosti (engl. *Locked in sindrom*) unos teksta i interakciju s računalom [1].

Postoji nekoliko različitih implementacija unosa teksta pomoću praćenja pomaka oka. Najčešći je sustav u kojem korisnik gleda u slovo na ekranu koje želi odabrati. Za rad ovakvog sustava potreban je specijalni uređaj za praćenje pomaka oka poput Tobii uređaja za praćenje očiju. Drugi često korišteni sustav temelji se na treptajima. U tom sustavu, aplikacija postepeno prolazi kroz redove na tipkovnici te korisnik treptajem odabire red nakon čega aplikacija iterira kroz slova u odabranom redu. Također, u jednom od često korištenih sustava, korisnik pomicanjem oka upravlja pokazivačem. Za odabir slova korisnik mora zadržati pokazivač na određenom slovu dovoljno dugo vremena.

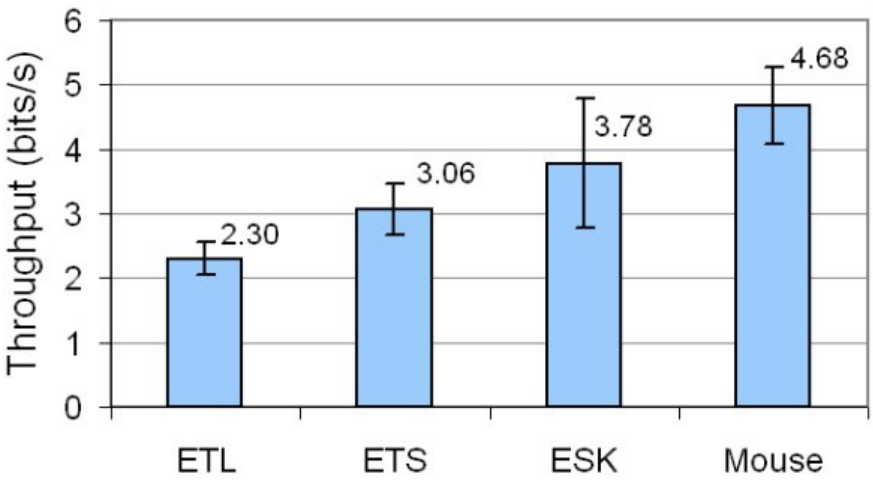
Najčešći problem s kojim se susreću svi sustavi zasnovani na praćenju pomaka oka je problem Midin dodira (engl. *Midas touch*). U ovom problemu, gdje god korisnik pogleda, to bude odabrano odnosno kliknuto. Još neki od čestih problema su mala brzina unosa teksta, visoka nepreciznost i otežano prepoznavanje pogleda u bilo kojim uvjetima.

Unos teksta zasnovan na praćenju pomaka oka je česta tema u interakciji čovjeka i računala. Jedan od poznatijih radova je "BlinkWrite: efficient text entry using eye blinks" [2], autora Scotta MacKenzieja i Behrooz Ashtianija. BlinkWrite sustav omogućava korisnicima unos i ispravak teksta koristeći samo treptaje. Na vrhu korisničkog sučelja BlinkWrite sustava (Slika 2.1) nalazi se polje koje prikazuje tekst koji korisnik treba unijeti te polje u koje korisnik unosi taj tekst. Nakon toga slijedi dio za odabir slova koji se sastoji od 4 grupe. Slova su podijeljena u 3 grupe dok četvrtu grupu čini tipka "Space". BlinkWrite sustav prolazi kroz sve grupe te korisnik treptajem odabire jednu grupe nakon čega sustav prolazi kroz sva slova u odabranoj grupi. U narednom redu nalazi se polje koje prikazuje trenutni slijed tipki. U zadnjem redu nalazi se lista kandidata koja sadrži riječi koje sustav dinamički generira na temelju već unesenih slova i riječi. Sustav iterira kroz sve riječi u listi kandidata te korisnik treptajem odabire jednu riječ što značajno povećava brzinu unosa teksta. Ovaj sustav razlikuje namjerne treptaje od refleksnih i prirodnih treptaja.



Slika 2.1 *Korisničko sučelje BlinkWrite sustava [2]*

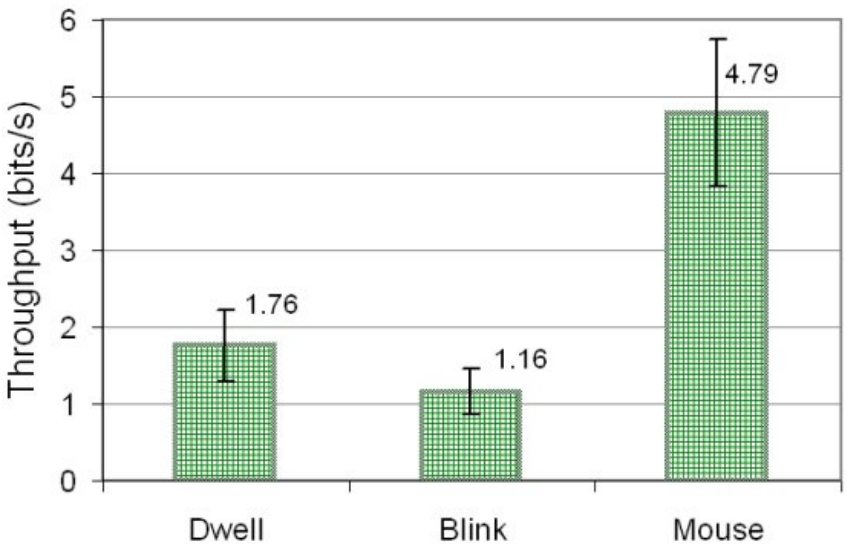
Još jedan poznat rad je "Evaluating Eye Tracking Systems for Computer Input" [3] čiji je autor isto Scott Mackenzie. Taj rad istražuje i uspoređuje različite sustave za unos temeljene na praćenju pogleda. U radu je provedeno dva eksperimenta. U prvom eksperimentu uspoređuju se dva modaliteta interakcije: zadržavanje pogleda (engl. *Dwell time*) s vremenom od 750 ms i 500 ms, te pritisak tipke (tipka "Space") za akcije pokazivanja i odabira meta koristeći Arrington Research ViewPoint sustav za praćenje pogleda. Rezultati za propusnost su prikazani na slici 2.2 gdje ETL označava vrijeme zadržavanja od 750 ms, ETS vrijeme od 500 ms, dok ESK predstavlja pritisak tipke. Modalitet interakcije s pritiskom tipke pokazao je najveću propusnost, dok je modalitet s vremenom zadržavanja od 750 ms imao najnižu propusnost. Modalitet interakcije s pritiskom tipke je pokazao najveću propusnost jer eliminira potrebno vrijeme zadržavanja za odabir mete.



Slika 2.2 *Propusnost za sve modalitete interakcije [3]*

U drugom eksperimentu uspoređuje se vrijeme zadržavanje i treptanje za akcije pokazivanja i odabira meta koristeći EyeTech Digital Systems3 TM3 sustav za praćenje pogleda. Za treptaj i vrijeme zadržavanja, trajanje je postavljeno na 500 ms. Grafička usporedba rezultata prikazana je na slici 2.3. Na grafu "Dwell" predstavlja vrijeme zadržavanja dok "Blink" predstavlja treptaj. Propusnost za vrijeme zadržavanja bila je značajno veća u odnosu na treptaj. Rezultati za propusnost su znatno

niži u usporedbi s rezultatima iz prvog eksperimenta. Razlog tome leži u različitim sustavima praćenja pogleda koji su korišteni, kao i u znatno manjoj širini i većoj udaljenosti meta koju je korisnik trebao odabrati.



Slika 2.3 *Propusnost za sve modalitete interakcije [3]*

Poglavlje 3

Razvoj metoda za unos teksta zasnovanih na praćenju pomaka oka

Sustav za unos teksta temeljen na praćenju pomaka oka implementiran je za operacijski sustav Windows 10. Za implementaciju modaliteta interakcije i virtualnih tipkovnica korišten je programski jezik Python. Python je također korišten za prediktivno modeliranje i analizu eksperimentalnih rezultata. Uz Python, u analizi eksperimentalnih rezultata korišten je i program paket SPSS kojeg je razvio IBM.

3.1 Tehnološki stog

U ovome poglavlju opisan je tehnološki stog koji je korišten za izradu spomenutih modaliteta interakcije i virtualnih tipkovnica.

3.1.1 Python

Python je programski jezik visoke razine koji je poznat po svojoj jednostavnosti i čitljivosti [4]. Ovo ga čini idealnim za brzi razvoj aplikacija. Python je stvorio Guido van Rossum 1990. godine. Python omogućuje korištenje različitih stilova programiranja, uključujući objektno-orijentirano, strukturno i aspektno orijentirano programiranje. Osim toga, Python nudi veliku količinu knjižnica koje omogućuju

jednostavnu implementaciju različitih aplikacija. U nastavku je dan opis pojedinih knjižnica koje su korištene tijekom razvoja.

Za implementaciju praćenja pomaka oka korištena je knjižnica `GazeTracking` [5], koju je razvio Antoine Lamé. Ova knjižnica omogućava praćenje pogleda korisnika pomoću web kamere. Pruža informacije o položaju zjenica i smjeru pogleda, uključujući lijevo i desno. Za detekciju očiju, `GazeTracking` koristi detektor oznaka na licu iz `Dlib` knjižnice. Nadalje, još jedna bitna funkcionalnost knjižnice je detekcija treptaja što je u aplikaciji korišteno za simulaciju klika mišem. Osnovna funkcionalnost ove knjižnice je detekcija lica i očiju. Za detekciju lica i očiju korištene su napredne metode računalnog vida i obrade slika. `GazeTracking` nudi jasno definirane i intuitivne metode koje programerima omogućuju brzu implementaciju funkcionalnosti praćenja pogleda.

`PyAutoGUI`[6] je knjižnica koja omogućuje Python skriptama kontrolu nad mišem i tipkovnicom. Knjižnica omogućuje simuliranje radnji poput pomicanja i klikanja miša te tipkanja na tipkovnici. Dodatno, `PyAutoGUI` omogućava snimanje snimki zaslona i prepoznavanje položaja specifičnih elemenata na zaslonu na temelju dane slike. Također, knjižnica može automatizirati zadatke poput ispunjavanja obrazaca te locirati i pomicati prozor aplikacije. U modalitetu slobodnog kretanja `PyAutoGUI` je korišten za pomicanje pokazivača sukladno promjeni položaja zjenica te za simulaciju klika mišem prilikom detekcije treptaja. Knjižnica je kompatibilna s operacijskim sustavima Windows, macOS i Linux te ujedno Python 2 i Python 3 verzijama.

Virtualne tipkovnice napravljene su u Python knjižnici `tkinter` [7]. `Tkinter` je standardna Python knjižnica za izgradnju grafičkih korisničkih sučelja (engl. `Graphical User Interface`, GUI). Knjižnica nudi različite alate za kreiranje aplikacija, uključujući okvire (engl, *frames*), gumbe (engl. *buttons*), polja za unos (engl. *input fields*) i oznake (engl. *tags*). Dodatno, `tkinter` omogućava organizaciju elemenata unutar prozora pomoću različitih upravitelja rasporeda poput mreže (engl. *grid*) i pakiranja (engl. *pack*).

`Keyboard` [8] je jednostavna knjižnica koja omogućava potpunu kontrolu nad tipkovnicom u aplikaciji. Ova knjižnica omogućuje slušanje svih događaja s tipkovnice te slanje vlastitih događaja. Nadalje, omogućuje i simuliranje pritiska pojedinih tipki tipkovnice. Knjižnica je kompatibilna s operacijskim sustavima Windows i Li-

nux, te podržava Python verzije 2 i 3. U aplikaciji, `keyboard` knjižnica se koristi za zaustavljanje rada pritiskom na tipku 'Esc'.

Za upravljanje pristupom kameri koristi se modul cv2. Ovaj modul je dio knjižnice `OpenCV` [9] i koristi se za obradu slika i videa u Python-u. Omogućuje niz funkcionalnosti poput: manipulacije slika, filtriranja slika, detekcije objekata, detekcije pokreta i prepoznavanja lica. Također, podržava razvoj modela strojnog učenja (engl. *machine learning*) i dubokog učenja (engl. *deep learning*) koji se koriste u klasifikaciji i segmentaciji slika. Korištenjem metode `VideoCapture()` otvara se prijenos sa zadane kamere što omogućuje prikupljanje video okvira.

`Pynput` [10] je knjižnica koja omogućava praćenje i simuliranje unosa putem tipkovnice i miša. Sadrži potpaket za svaki podržani uređaj: `pynput.mouse` i `pynput.keyboard`. Korištenjem razreda `Key` iz potpaketa `pynput.keyboard` definiraju se specijalne tipke na tipke na tipkovnici: 'Enter', 'Backspace', 'Space' i 'Caps Lock' dok se korištenjem klase `Controller` simulira pritisak definiranih specijalnih tipki.

Knjižnica `time` [11] koja je dio Python instalacije pruža razne funkcije vezane uz vrijeme. Omogućuje rad s vremenskim intervalima, datumima i vremenima. U aplikaciji se `time` koristi za upravljanje i mjerenje vremena koje korisnik provodi gledajući u određene smjerove ili trepćući.

U aplikaciji, knjižnica `threading` [12] koristi se za pokretanje metode za praćenje pomaka očiju u zasebnoj niti (engl. *thread*). Ovaj način rada omogućava održavanje responzivnosti grafičkog korisničkog sučelja jer metoda za praćenje pomaka očiju može biti vrlo zahtjevna za procesor. Knjižnica `threading` omogućuje istovremeno izvršavanje više zadataka unutar iste aplikacije. Paralelno izvršavanje zadataka znatno ubrzava izvođenje programa te povećava iskoristivost resursa.

`Multiprocessing` [13] knjižnica omogućuje stvaranje novih procesa koji se izvode paralelno, neovisno jedan o drugome. Svaki proces ima vlastitu memoriju i resurse. To omogućuje bolje iskorištavanje višejezgrenih kapaciteta procesora. Knjižnica koriste različite mehanizme za komunikaciju između procesa. Mehanizmi za komunikaciju uključuju redove (engl.*queue*), polja (engl. *array*) te kanale (engl. *pipe*). Za sinkronizaciju dostupni su mehanizmi poput uvjeta (engl. *condition*) i brava (engl. *lock*). Aplikacija koristi metodu `Proces` kako bi omogućila da rad s kamerom,

upravljanje tipkovnicom i upravljanje pokazivačem za modalitet slobodnog kretanja odvijaju paralelno i neovisno jedan o drugome. Za modalitet unosa lijevo-desno, svi zadaci odvijaju se u jednom procesu, jer je logika unosa lijevo-desno usko povezana s tipkovnicom.

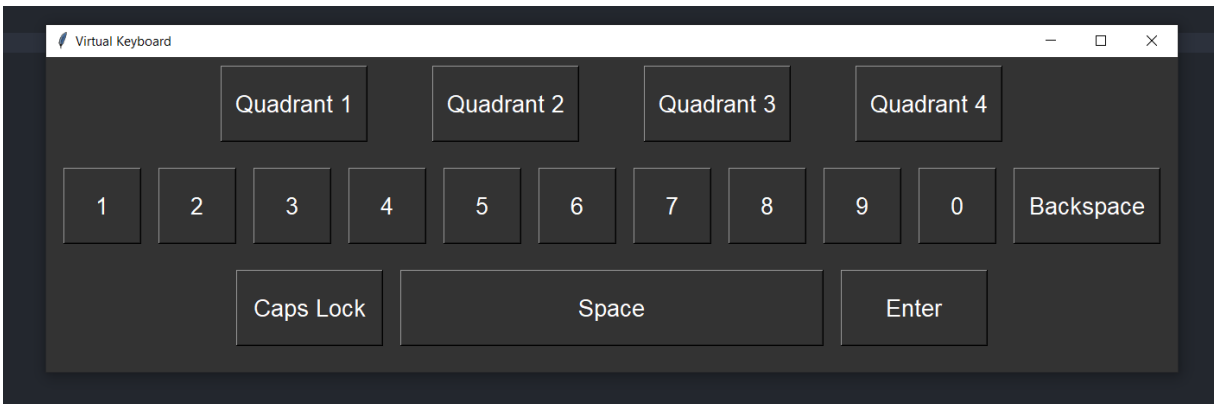
3.2 Dizajn tipkovnica

Implementirane su dvije virtualne tipkovnice. Prva virtualna tipkovnica podijeljena je u kvadrante, pri čemu svaki kvadrant sadrži jedan red slova ili brojeva s klasične QWERTY tipkovnice. Druga virtualna tipkovnica predstavlja klasičnu QWERTY tipkovnicu.

3.2.1 Tipkovnica s kvadrantima

Kao što samo ime sugerira tipkovnica je podijeljena na 4 glavna kvadranta (Slika 3.1). U prvom redu nalaze se 4 tipke za odabir željenog kvadranta. Drugi red sadrži slova i brojeve, koji se dinamički mijenjaju ovisno o trenutno odabranom kvadrantu. Prilikom pokretanja aplikacije automatski je odabran prvi kvadrant. Osim standardnih slova i brojeva, tipkovnica uključuje i specijalne tipke kao što su "Backspace", "Enter", "Caps Lock" i "Space". Ove specijalne tipke nalaze se na donjem dijelu ekrana u trećem redu, osim tipke "Backspace" koja se nalazi na kraju drugog reda. Za razliku od slova u drugom redu specijalne tipke su uvijek vidljive na tipkovnici neovisno o odabranom kvadrantu. Svaki kvadrant sadrži jedan red znakova s klasične QWERTY tipkovnice:

- **Prvi kvadrant**: sadrži numeričke tipke (0-9), omogućavajući korisnicima unos brojeva;
- **Drugi kvadrant**: obuhvaća gornji red slova (QWERTYUIOP);
- **Treći kvadrant**: uključuje srednji red slova (ASDFGHJKL?) i znak '?';
- **Četvrti kvadrant**: sadrži donji red slova (ZXCVBNM) i nekoliko specijalnih znakova (,.!);



Slika 3.1 *Tipkovnica s kvadrantima*

Inicijalizacija varijabli

Na početku programskog koda definirane su glavne varijable koje se koriste u ostatku koda. Varijabla `keyboard_layouts` predstavlja raspored znakova po kvadrantima, dok `current_quadrant` predstavlja trenutno odabrani kvadrant. Za simulaciju pritiska tipki koristi se varijabla `kb` koja je objekt razreda `Controller` iz *Pynput* biblioteke. Varijabla `highlighted_key` pohranjuje referencu na trenutno istaknutu tipku, dok `quadrant_buttons` sadrži gumbe za odabir kvadranta.

Proces stvaranja tipkovnice

Proces stvaranja tipkovnice sastoji se od dvije funkcije: `add_quadrant_buttons` (Ispis 3.1) i `refresh_keyboard` (Ispis 3.2). U prvoj funkciji (Ispis 3.1) dodaju se gumbi za odabir kvadranta na glavno sučelje aplikacije. Ova funkcija iterira kroz listu `keyboard_layouts` te za svaki red stvara jedan gumb za kvadrant. Svakom gumbu dodjeljuje se metoda `on_quadrant_click`, koja kao argument prima redni broj kvadranta.

```
1 def add_quadrant_buttons():
2     global quadrant_buttons
3     for i, quadrant in enumerate(keyboard_layouts):
4         quadrant_button=tk.Button(
5         root, text=f"Quadrant {i+1}", width=10, height=2,
```

```
        command=lambda q=i:on_quadrant_click(q),font=('Arial', 16))

        quadrant_button.grid(row=1, column=i+3, padx=10, pady=10)
        quadrant_buttons.append(quadrant_button)
```

Ispis 3.1 Implementacija metode za dodavanje gumba na glavno sučelje aplikacije

Kada ispitanik odabere ciljani kvadrant, potrebno je ažurirati znakove na tipkovnici sa znakovima iz trenutno odabranoga kvadranta. Za ovaj slučaj, koristi se funkcija `refresh_keyboard` koja briše prethodne elemente s prikaza i dodaje nove tipke. Na početku, metoda iterira kroz sve elemente u `keyboard_frame` varijabli i pozivom `destroy` funkcije uklanja sve elemente. Nakon uklanjanja starih elemenata, dohvaća se novi raspored tipki iz `keyboard_layouts` koristeći varijablu `current_quadrant` te se sprema u varijablu `row_layout`. Slijedi iteracija kroz redove tipki gdje svaki `key_row` predstavlja jedan red, a unutar svakog reda se generiraju tipke za svaki znak pomoću funkcije `create_button`. Funkcija kao parametre prima: okvir kojem tipka pripada, tekst tipke, veličinu, font te funkciju `on_key_press` koja se poziva prilikom pritiska tipke. Nakon postavljanja tipki specifičnih za trenutni kvadrant, na kraju reda dodaje se tipka 'Backspace' koja je stvorena pomoću iste funkcije `create_button`. Tipki se dodjeljuje veća širina kako bi bila upečatljivija. Nakon dodavanja 'Backspace' tipke inkrementira se 'row' indeks kako bi se prešlo u zadnji red. Zadnji red sadrži već prije spomenute specijalne znakove: 'Caps Lock', 'Enter', 'Space' koji se dodaju u zaseban okvir `bottom_row_frame`. Svaka specijalna tipka stvara se uz pomoć `special_keys` rječnika koji sadrži ime i širinu tipke te funkcije `create_button`.

```
def refresh_keyboard():
    global bottom_row_frame
    for widget in keyboard_frame.winfo_children():
        widget.destroy()

    row_layout = keyboard_layouts[current_quadrant]
    row = 0
    for key_row in row_layout:
        col = 0
        for key in key_row:
            if key != " ":
```

```
                key_button = create_button(
                keyboard_frame, key, 5, 2,
                lambda k=key: on_key_press(k), ('Arial', 16)
                )

                key_button.grid(row=row, column=col, padx=2, pady=2)
            col += 1

        backspace_button = create_button(
        keyboard_frame, "Backspace", 10, 2,
        lambda k="Backspace": on_key_press(k), ('Arial', 16)
        )

        backspace_button.grid(row=row, column=col, columnspan=10,
        padx=10, pady=10)
        row += 1

    bottom_row_frame = tk.Frame(keyboard_frame)
    bottom_row_frame.grid(row=row, column=0, columnspan=col,
    padx=10, pady=10)

    left_label = tk.Label(bottom_row_frame, width=23)
    left_label.pack(side=tk.LEFT, expand=True)

    special_keys = {"Caps Lock": 10, "Space": 30, "Enter": 10}
    for key, width in special_keys.items():
        button = create_button(
        bottom_row_frame, key, width, 2,
        lambda k=key: on_key_press(k), ('Arial', 16)
        )
        button.pack(side=tk.LEFT, padx=10, pady=10)
```

Ispis 3.2 Metoda za osvježavanje tipkovnice

Prilikom promjene veličine prozora, može doći do narušavanja rasporeda elemenata. Kako bi se to izbjeglo koristi se funkcija `configure_weights` (Ispis 3.3) koja konfigurira raspored elemenata aplikacije, omogućavajući elementima da se ravnomjerno proširuju i popune dostupni prostor. Postavljanjem težine redaka i stupaca, osigurava se da elementi sučelja ostanu proporcionalni i da se pravilno prilagođavaju

promjenama veličine prozora.

```
def configure_weights():
    for i in range(10):
        root.grid_rowconfigure(i, weight=3)
        root.grid_columnconfigure(i, weight=3)
```

Ispis 3.3 Metoda za automatsko re-konfiguriranje rasporeda tipki

Proces simulacije pritiska tipke

Proces za simulaciju pritiska tipke definiran je funkcijom `on_key_press` (Ispis 3.4). Funkcija kao parametar prima `key` koji definira koju tipku je potrebno pritisnuti. Rječnik `special_keys` koristi se mapiranje imena specijalnih tipki na odgovarajuće akcije. Za kvadrant tipke poziva se `on_quadrant_click` metoda dok se ostale specijalne tipke definiraju koristeći `Key` razred iz *Pynput* knjižnice. Funkcija prvo provjerava je li parametar `key` prisutan u rječniku `special_keys` i ,ako je prisutan, izvodi se akcija koja je pridružena toj tipki. U suprotnom, simulira se pritisak tipke kao običnog znaka koristeći varijablu `kb`.

```
def on_key_press(key):
    root.withdraw()
    special_keys = {
        "Backspace": Key.backspace,
        "Enter": Key.enter,
        "Space": Key.space,
        "Caps Lock": Key.caps_lock,
        "Quadrant 1": lambda: on_quadrant_click(0),
        "Quadrant 2": lambda: on_quadrant_click(1),
        "Quadrant 3": lambda: on_quadrant_click(2),
        "Quadrant 4": lambda: on_quadrant_click(3)
    }
    if key in special_keys:
        action = special_keys[key]
        if callable(action):
            root.after(10, action)

```

```
18          else:
19              root.after(10, kb.tap(action))
20
21      else:
22          root.after(10, kb.tap(key))
23      root.deiconify()
```

Ispis 3.4 Metoda za simulaciju pritiska tipke

Kada ispitanik aktivira jednu od tipki za odabir kvadranta, poziva se funkcija `on_quadrant_click` (Ispis 3.5). Ona postavlja globalnu varijablu `current_quadrant` na vrijednost odabranoga kvadranta i poziva funkciju `refresh_keyboard` kako bi ažurirala prikaz tipki na sučelju tipkovnice. Promjenom vrijednosti `current_quadrant` varijable dinamički se mijenjaju tipke u drugom redu tipkovnice.

```
1 def on_quadrant_click(quadrant):
2     global current_quadrant, highlighted_key
3     current_quadrant = quadrant
4     refresh_keyboard()
```

Ispis 3.5 Metoda za aktivaciju ciljanog kvadranta

Glavna metoda

U `main` funkciji (Ispis 3.6) inicijalizira se glavni *Tkinter* prozor, postavlja se njegov naslov te osigurava da je prozor prilikom pokretanja aplikacije na vrhu drugih prozora. Također konfigurira se tamna tema aplikacije pomoću `tk_setPalette`. Nadalje, unutar glavnog *Tkinter* prozora stvara se okvir `keyboard_frame` koji sadrži tipke specifične za svaki kvadrant. Okvir se pozicionira pomoću funkcije `grid`, pri čemu se početni redak postavlja na 2, a početni stupac na 0. Na samom kraju, pozivaju se metode za stvaranje tipkovnice, dodavanje gumba za odabir kvadranta te se pokreće glavna petlja.

```
def main():
    global root, quadrant_buttons, keyboard_frame

    root = tk.Tk()
    root.attributes('-topmost', True)
    root.title("Virtual Keyboard")

    root.tk_setPalette(background='#333', foreground='#fff',
    activeBackground='#444', activeForeground='#fff')

    add_quadrant_buttons()
    configure_weights()

    keyboard_frame = tk.Frame(root)
    keyboard_frame.grid(row=2, column=0, columnspan=10, padx=10,
    pady=10)
    refresh_keyboard()

    root.mainloop()
```

Ispis 3.6 Metoda za inicijalizaciju tipkovnice s kvadrantima

3.2.2 QWERTY Tipkovnica

Kao druga virtualna tipkovnica implementirana je standardna QWERTY tipkovnica (Slika 3.2). Na lijevoj strani nalaze se slova i brojke čiji raspored odgovara klasičnom QWERTY rasporedu, dok se na desnoj strani nalaze specijalne tipke poput 'Backspace', 'Enter' i 'Caps Lock'. Na samom dnu tipkovnice nalazi se 'Space' tipka. Ova tipkovnica ima veće tipke u usporedbi s tipkovnicom s kvadrantima što je potrebno radi bolje preglednosti jer su sve tipke odmah dostupne na tipkovnici.

Slika 3.2 *QWERTY tipkovnica*

Inicijalizacija varijabli

Na početku, uvoze se sve potrebne knjižnice. Nakon toga definira se metoda `run_ons_keys` koja pokreće cijeli programski kod (Ispis 3.7). Za simulaciju pritiska tipke u ovom kodu isto se koristi varijabla `kb`. Nadalje, varijabla `keyboard_layouts` kao i u tipkovnici s kvadrantima definira raspored tipki na tipkovnici, podijeljenih u redove. Svaki element u ovom polju je lista koja predstavlja jedan red tipki, s posebnim oznakama za specijalne tipke: "Backspace", "Enter" i "Caps Lock".

```
1  import tkinter as tk
2  from tkinter import *
3  from pynput.keyboard import Key, Controller
4
5  class KeyB(object):
6
7      def run_ons_key():
8
9          kb = Controller()
10
```

```
        keyboard_layout = [
            ["1", "2", "3", "4", "5", "6", "7", "8", "9", "0",
            "Backspace"],
            ["q", "w", "e", "r", "t", "y", "u", "i", "o", "p",
            "Enter"],
            ["a", "s", "d", "f", "g", "h", "j", "k", "l", "?"],
            ["z", "x", "c", "v", "b", "n", "m", ",", ".", "!",
            "Caps Lock"]
        ]
```

Ispis 3.7 Inicijalizacija programskog koda

Proces simulacije pritiska tipke

Način simulacije pritiska tipke (Ispis 3.8) za QWERTY tipkovnicu sličan je kao i u tipkovnici s kvadrantima (Ispis 3.4), ali je jednostavniji. Funkcija kao ulazni parametar prima varijablu `key` koja određuje koju tipku je potrebno pritisnuti. Najprije se provjerava da li se parametar `key` nalazi u rječniku `special_key` te ,ako je prisutan, izvršava se odgovarajuća akcija. U suprotnom, izvršava se simulacija pritiska tipke koristeći varijablu `kb`. Glavna razlika između ove implementacije i one u tipkovnici s kvadrantima je u tome što rječnik ne sadrži tipke za kvadrante.

```
def on_key_press(key):
    root.withdraw()
    special_keys = {
        "Backspace": Key.backspace,
        "Enter": Key.enter,
        "Space": Key.space,
        "Caps Lock": Key.caps_lock,
    }

    if key in special_keys:
        action = special_keys[key]
        if callable(action):
            root.after(10, action)
        else:
            root.after(10, kb.tap(action))

```

```
    else:
        root.after(10, kb.tap(key))
    root.deiconify()
```

Ispis 3.8 Metoda simulacije pritiska tipke na QWERTY tipkovnici

Proces stvaranja QWERTY tipkovnice

Proces stvaranja QWERTY tipkovnice je sličan načinu stvaranja tipkovnice s kvadrantima. Funkcija `refresh_keyboard` (Ispis 3.9) na početku definira konstante za visinu i širinu tipku. Nakon toga, koristeći *for* petlju iterira kroz svaki red i tipku u `keyboard_layout`-u. Pri svakoj iteraciji petlje, provjerava je li `key` specijalna tipka. Ako `key` jest specijalna tipka, dodaje se u `specialframe` okvir u kojem tipke imaju veću širinu. Za sve ostale tipke, gumbi se stvaraju u `mainframe` okviru s normalnom širinom. Funkcija `on_key_press`, koja se izvršava prilikom pritiska tipke, dodjeljuje se svakoj tipki. Tipka za razmak ("Space") se stvara odvojeno, s većom širinom kako bi bila lakše dostupna.Na samom kraju, stupci i retci tipkovnice se centriraju koristeći `grid_rowconfigure` i `grid_columnconfigure funkcije`.

```
def init_keyboard():
    button_width = 6
    button_height = 3

    for row, row_keys in enumerate(keyboard_layouts):
        for col, key in enumerate(row_keys):
            if key == "Backspace" or key == "Enter"
            or key == "Caps Lock":
                key_button = tk.Button(
                specialFrame,text=key, width=button_width*4,
                height=button_height,
                command=lambda k=key: on_key_press(k),
                font=('Arial', 16)
                )
                key_button.grid(row=row, column=0, sticky="nsew")
            else:
                key_button = tk.Button(
                mainFrame, text=key, width=button_width,
                height=button_height,
```

```
                command=lambda k=key: on_key_press(k),
                font=('Arial', 16)
                )
                key_button.grid(row=row, column=col, sticky="nsew")

    key_sp = tk.Button(
    keyboard_frame, text="Space", width=button_width*11,
    height=button_height,
    command=lambda k="Space": on_key_press(k), font=('Arial', 16)
    )
    key_sp.grid(row=1,column=1)

    for i in range(len(keyboard_layouts) + 1):
        root.grid_rowconfigure(i, weight=1)
    for i in range(11):
        root.grid_columnconfigure(i, weight=1)
```

Ispis 3.9 Metoda za inicijalizaciju QWERTY tipkovnice

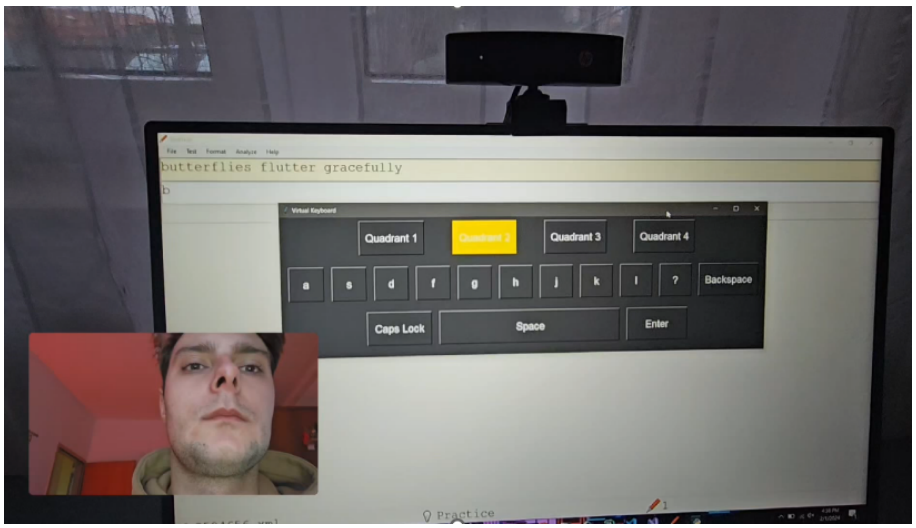
3.3 Modaliteti interakcije

Razvijena su dva modaliteta unosa teksta pomoću pomaka oka. Prva metoda koristi praćenje pokreta oka za kontrolu pokazivača, dok druga metoda koristi isticanje trenutnog slova na tipkovnici žutom bojom, pri čemu ispitanik pogledom lijevo ili desno odabire željeno susjedno slovo. U obje metode treptaj se koristi za simulaciju odabira.

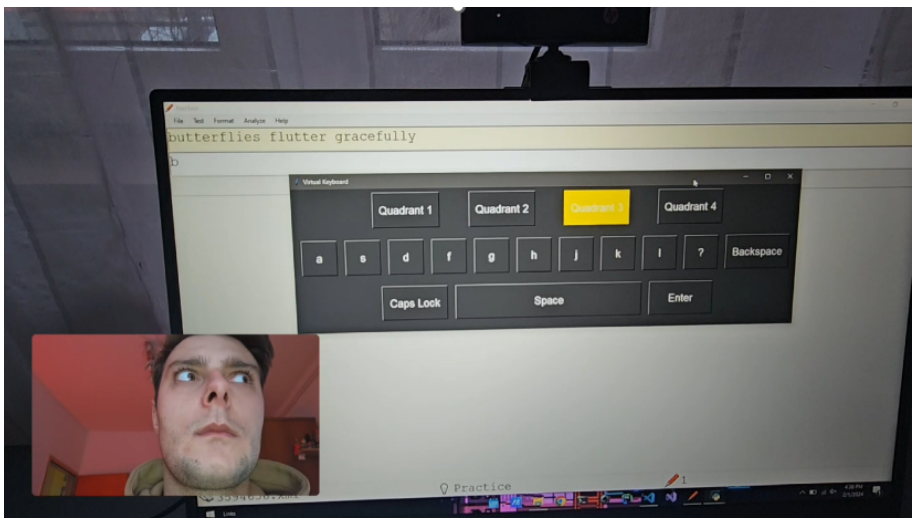
3.3.1 Pomak lijevo-desno

Prvi modalitet unosa koji je implementiran je modalitet lijevo-desno. U ovom modalitetu trenutno slovo je istaknuto žutom bojom te korisnik pogledom u lijevu ili desnu stranu odabire sljedeće slovo. Na slici 3.3 može se vidjeti da kada korisnik gleda ravno u kameru odabrana je tipka "Quadrant 2", dok na slici 3.4 može se vidjeti da je korisnik pogledom u desnu stranu prešao na tipku "Quadrant 3". Za odabir odnosno simulaciju klika tipke koristi se detekcija treptanja. Ako bi se za si-

mulaciju klika koristila svaka detekcija treptanja namjerna ili refleksna, to bi dovelo do velikog broja pogrešaka. Iz tog razloga postavljen je vremenski prag koliko dugo korisnik mora imati zatvorene oči da se simulira klik tipke.



Slika 3.3 *Primjer korištenja aplikacije kada korisnik gleda ravno u kameru*



Slika 3.4 *Primjer korištenja aplikacije kada korisnik gleda u stranu*

Proces prepoznavanje pogleda

Najvažniji dio modaliteta lijevo-desno je provjera u kojem smjeru korisnik gleda i detekcija treptaja, kao i mjerenje koliko dugo traju te radnje. Ovaj problem rješava se funkcijom `check_time_looking` (Ispis 3.10, Ispis 3.11). Kao parametre funkcija prima: `gaze` koji je instanca razreda `GazeTracking`, `webcam` instancu razreda `VideoCapture` i `threshold seconds` koji definira vremenski prag za pogled u lijevo ili desno. Na početku, funkcija definira rječnik `states` koji ima tri moguća stanja: *left, right* i *blinking*. Svako stanje sadrži početno vrijeme `start_time`, proteklo vrijeme `elapsed_time`, akciju koju je potrebno izvršiti (`action`) i `message` koji se ispisuje za određenu akciju. Uz navedeno, stanje `blinking` dodatno sadrži i `threshold`.

```
def check_time_looking(gaze, webcam, threshold_seconds=0.6):
    states = {
        "left": {"start_time": None, "elapsed_time": 0,
        "action": move_left, "message": "User has been looking left
        for 0.6 seconds"},
        "right": {"start_time": None, "elapsed_time": 0,
        "action": move_right, "message": "User has been looking
        right for 0.6 seconds"},
        "blinking": {"start_time": None, "elapsed_time": 0,
        "action": press_current_key, "message": "User has been
        blinking for 2 seconds", "threshold": 1.5}
    }
```

Ispis 3.10 Metoda za praćenje pogleda (1)

Unutar beskonačne *while* petlje konstantno se učitava novi okvir iz kamere te se osvježavaju podatci o pogledu na temelju trenutnog okvira (Ispis 3.11). Korištenjem *for* petlje iterira se kroz sva stanja u `states` rječniku te se provjerava svako definirano stanje. Ako je određeno stanje detektirano, prvo se vrijednosti za početno vrijeme u ostalim stanjima resetira. Nakon toga, vrši se provjera da li detektirano stanje ima postavljeno početno vrijeme. Ako početno vrijeme ne postoji, onda se postavi na trenutno vrijeme, u suprotnom proteklo vrijeme se postavi na razliku trenutnog vremena i početnog vremena. Na kraju, vrši se provjera je li proteklo vrijeme veće od zadanog praga, ispisuje se poruka, izvršava se odgovarajuća akcija te se ponovno

postavlja početno vrijeme.

```
    while gaze_thread_running:
        _, frame = webcam.read()
        gaze.refresh(frame)
        for state, info in states.items():
            if getattr(gaze, f"is_{state}")():
                for other_state in states:
                    if other_state != state:
                        states[other_state]["start_time"] = None

                if info["start_time"] is None:
                    info["start_time"] = time.time()

                else:
                    info["elapsed_time"] =
                    time.time() - info["start_time"]
                if info["elapsed_time"] >=
                info.get("threshold", threshold_seconds):
                    print(info["message"])
                    info["action"]()
                    info["start_time"] = time.time()

            else:
                info["start_time"] = None
    webcam.release()
```

Ispis 3.11 Metoda za praćenje pogleda (2)

Proces odabira istaknutog slova

Kada funkcija `check_time_looking` (Ispis 3.10, 3.11) detektira da je pogled u lijevo ili desno trajao duže od zadanog praga, potrebno je istaknuti sljedeću tipku ovisno o smjeru pogleda. Ovaj slučaj rješava se uz pomoć funkcija `move_left` (Ispis 3.12) i `move_right` (Ispis 3.13). U funkciji `move_left` prvo se dohvaća položaj trenutno istaknute tipke uz pomoć globalne varijable `current_key_index`. Ako je indeks stupca veći od nule, pomiče se ulijevo smanjujući `col` za 1. U slučaju da je indeks stupca nula, a indeks reda je veći od nula, prelazi se u prethodni red u zadnji stupac.

Ako je trenutno istaknuta tipka u prvom stupcu prvog reda, prelazi se u zadnji stupac zadnjeg reda. Na kraju se poziva funkcija `select_key` kojoj se kao parametri prosljeđuju indeksi stupca i retka.

```
def move_left():
    global current_key_index
    row, col = current_key_index
    if col > 0:
        col -= 1
    elif row > 0:
        row -= 1
        col = len(all_keys[row]) - 1
    elif row == 0 and col == 0:
        row = len(all_keys) - 1
        col = len(all_keys[row]) - 1
    select_key((row, col))
```

Ispis 3.12 Metoda za označavanje tipke lijevo od trenutno aktivne

Logika u funkciji `move_right` je inverz logike u funkciji `move_left`. Nakon što se dohvati indeks stupca i retka, provjerava se je li indeks stupca izvan retka te ako je prelazi se u prvi stupac sljedećeg reda. U slučaju da je indeks red izvan granica tipkovnice prelazi se u prvi stupac prvog reda. Na kraju se poziva funkcija `select_key`.

```
def move_right():
    global current_key_index
    row, col = current_key_index
    col += 1

    if col >= len(all_keys[row]):
        col = 0
        row += 1

    if row >= len(all_keys):
        row = 0
        col = 0
    select_key((row, col))
```

Ispis 3.13 Metoda za označavanje tipke desno od trenutno aktivne

Primarna svrha funkcije `select_key` (Ispis 3.3.1) je označavanje trenutno odabrane tipke promjenom boje što je čini vizualno istaknutom. Funkcija započinje resetiranjem izgleda prijašnje istaknute tipke, vraćajući joj izvornu boju pozadine i teksta. Zatim se, pomoću indeksa stupca i retka koji su proslijeđeni funkciji kao parametri pronalazi nova tipka koju je potrebno označiti. Tipka se vizualno označava mijenjanjem pozadinske boje u žutu. Konačno, ažuriraju se globalne varijable `highlighted_key` i `current_key_index`.

```
def select_key(key_index):

    global highlighted_key, current_key_index

    if highlighted_key:
        highlighted_key.configure(bg='#333', fg='#fff')

    row, col = key_index
    key_button = all_keys[row][col]
    key_button.configure(bg="yellow")
    highlighted_key = key_button
    current_key_index = (row, col)
```

Na kraju, potrebno je spomenuti i funkciju `press_current_key` (Ispis 3.14) koja se poziva kada ispitanik trepće duže od zadanog vremenskog praga. Funkcija provjerava da li postoji `highlighted_key` te ako postoji poziva metodu `on_key_press` s tekstom označene tipke. Ovisno je li riječ o tipkovnici s kvadrantima ili QWERTY tipkovnici, poziva se odgovarajuća metoda za simulaciju pritiska.

```
def press_current_key():
    if highlighted_key:
        key = highlighted_key["text"]
        on_key_press(key)
```

Ispis 3.14 Metoda za simulaciju pritiska trenutno odabrane tipke

3.3.2 Slobodna putanja pokazivača

Drugi modalitet unosa koji je implementiran jest modalitet slobodnog kretanja pokazivača. U ovom modalitetu korisnik pomakom očiju pomiče sami pokazivač. Važno je napomenuti da korisnik ne usmjerava pokazivač gledanjem u određenu točku na ekranu. Umjesto toga, smjer pomicanja pokazivača određuje se na temelju razlike između početne i trenutne pozicije očiju u prostoru. Kao i u modalitetu lijevo-desno, za simulaciju klika koristi se detekcija treptanja.

Proces pomicanja pokazivača

Proces pomicanja pokazivača izvršava se u beskonačnoj *while* petlji (Ispis 3.15). Pri svakoj iteraciji, ažurira se `global_timer_end` kako bi se zabilježilo trenutno vrijeme. Ako je razlika između `global_timer_end` i `global_timer_start` veća od zadanog praga `blinking_interval_threshold`, varijabla `allow_blink` postavlja se na *true*. Uloga varijable `allow_blink` je spriječiti slučajno ili prekomjerno registriranje treptaja kao klikova. Svaka iteracija petlje dohvaća novi okvir iz web kamere, nakon čega se osvježavaju podatci o pogledu (engl. *gaze*) na temelju trenutnog okvira. Koordinate lijeve i desne zjenice dobivaju se pomoću metoda `pupil_left_coords` i `pupil_right_coords` iz trenutnog pogleda.

```
        while True:

            global_timer_end = time.time()

            if global_timer_end - global_timer_start >
            blinking_interval_threshold:

            allow_blink = True
            _, frame = webcam.read()

            gaze.refresh(frame)

            left_pupil = gaze.pupil_left_coords()
            right_pupil = gaze.pupil_right_coords()
```

Ispis 3.15 Proces pomicanja pokazivača (1)

U sljedećem koraku, vrši se provjera jesu li koordinate valjane. Ako su koordinate valjane, računa se središnja točka pogleda korisnika (`sum_x, sum_y`). U sljedećem koraku, sustav provjerava je li `sum_x` veći od granice `right_left_limit`. Ako jest, onda korisnik gleda u desno te je potrebno umanjiti `sum_x` za korak (engl. *step*), u suprotnom korisnik gleda u lijevo pa je potrebno povećati `sum_x` za korak. Ista provjera se vrši i za vertikalni smjer pokazivača. Ako je središnja točka pogleda ispod granice `up_down_limit`, pokazivač se pomiče gore i obrnuto. Na kraju se pokazivač miša premješta na nove koordinate pomoću `pyautogui.moveTo` funkcije.

```
            if left_pupil and right_pupil:
                sum_x = round((left_pupil[0] + right_pupil[0]) / 2)
                sum_y = round((left_pupil[1] + right_pupil[1]) / 2)

            if sum_x > right_left_lim:
                cursorX -= step

            elif sum_x < right_left_lim:
                cursorX += step

            if sum_y > up_down_lim:
                cursorY += step

            elif sum_y < up_down_lim:
                cursorY -= step

            pyautogui.moveTo(cursorX,cursorY)
```

Ispis 3.16 Proces pomicanja pokazivača (2)

Nadalje, na kraju petlje radi se detekcija treptaja (Ispis 3.17). Ako se detektira treptaj, te ako je varijabla `allow_blink` postavljena na *True*, ažurira se vrijeme `end`, a u suprotnom ažurira `start` vrijeme. U slučaju da je prošlo dovoljno vremena od zadnjeg treptaja (`blinking_lim`) program se pauzira na jednu sekundu, resetira se `start` vrijeme, postavi se `allow_blink` na *False*, ažurira se `global_timer_start` i simulira se klik mišem na trenutnim koordinatama pokazivača.

```
            if gaze.is_blinking() and allow_blink:
                end = time.time()
            else:
                start = time.time()
            if end - start >= blinking_lim:
                time.sleep(1)
                start = time.time()
                allow_blink = False
                global_timer_start = time.time()
                pyautogui.click(cursorX, cursorY)

            if(keyboard.is_pressed("esc")):
                break
        webcam.release()
        cv2.destroyAllWindows()
```

Ispis 3.17 Proces pomicanja pokazivača (3)

Poglavlje 4

Prediktivno i eksperimentalno vrednovanje

U ovom poglavlju opisan je postupak provedbe prediktivnog i eksperimentalnog vrednovanja. Na početku je dan opis Fittsovog zakona koji čini osnovu prediktivnog modeliranja u ovome radu. U sljedećem potpoglavlju opisan je dizajn HCI eksperimenta. Nakon toga opisane su aplikacije FittsStudy i TextTest koje su korištene za prikupljanje podataka u prediktivnom i eksperimentalnom vrednovanju kao i korištenih upitnika. Na samom kraju, dan je pregled same procedure provedbe eksperimenta.

4.1 Fittsov zakon i prediktivno modeliranje

Jedan od najpoznatijih prediktivnih modela u interakciji čovjeka i računala je Fittsov zakon [14]. Koristi se za modeliranje akcije pokazivanja i odabira poput pomaka prsta ili pokazivača do određenog cilja te odabir mete. Prema zakonu, vrijeme potrebno za brzo premještanje u ciljano području raste s udaljenošću te se smanjuje sa širinom cilja. U interakciji čovjeka i računala, Fittsov zakon se koristi u 3 svrhe: za provjeru da li uređaj ili metoda interakcije odgovaraju modelu, za analizu alternativnih dizajna putem prediktivnog modeliranja, te za korištenje indeksa izvedbe kao zavisne varijable u usporednoj analizi [14].

Fitts je predstavio svoj zakon u dva ključna rada, prvi objavljen 1954. godine,

a drugi 1964. godine. U svojim radovima Fitts je usporedio ljudske pokrete s informacijom u komunikacijskom kanalu [14]. Fitts je zaključio da ljudski operater koji izvodi pokret preko određene amplitude, kako bi dosegnuo metu određene širine ili tolerancije, pokazuje 'stopu prijenosa informacija' [15]. Iz tog razloga, razvoj Fittsovog zakona krenuo je od Shannon-Hartley teorema u kojem je informacijski kapacitet kanala C opisan sljedećim izrazom:

$$C = B \cdot \log_2 \left(1 + \frac{S}{N}\right) \tag{4.1}$$

gdje B označava širinu pojasa dok S označava snagu signala, a N predstavlja šum signala. U svom radu iz 1954. godine, Fitts je uveo novu metriku temeljenu na Shannon-Hartleyjevu teoremu, nazvanu indeks složenosti zadatka (ID, izražen u bitovima), definiranu sljedećim izrazom:

$$ID = \log_2 \left(\frac{2A}{W}\right) \tag{4.2}$$

Prema informacijskoj analogiji u izrazu za ID, udaljenost do središta cilja (A) predstavlja signal, dok širina cilja (W) predstavlja šum signala. Iz izraza za ID može se uočiti odnos udaljenosti do cilja i širine cilja. Konkretno, kada se udaljenost do cilja udvostruči, to ima isti efekt kao i kada se širina cilja smanji za pola.

Za poboljšanje informacijske analogije, Scott MacKenzie uveo je novu formulaciju indeksa složenosti zadatka 1992. godine [14]. Ova formulacija, zbog svoje sličnosti sa Shannon-Hartleyevim teoremom, naziva se Shannonova formulacija. Korištenjem ove formulacije indeks složenosti zadatka izjednačen je s količinom prenesene informacije (u bitovima) prilikom izvođenja zadatka. Formulacija je opisana sljedećim izrazom:

$$ID = \log_2 \left(\frac{A}{W} + 1\right) \tag{4.3}$$

Dodatno, Fitts je uveo indeks izvedbe (IP, bits/s) kao omjer indeksa složenosti zadataka (ID) i vremena pokreta (MT, u sekundama):

$$IP = \left(\frac{ID}{MT}\right) \tag{4.4}$$

Ovaj pokazatelj, poznat i kao propusnost (TP), koristi se za mjerenje ljudskih sposobnosti u izvođenju zadataka pokazivanja-i-odabira. U izrazu za TP, ID predstavlja preciznost ljudskog pokreta, dok MT odražava brzinu izvođenja pokreta.

Za izračun MT koristi se jednostavna linearna jednadžba:

$$MT = a + b \times ID \tag{4.5}$$

U ovoj jednadžbi, parametri a i b određuju se empirijskim testovima, najčešće primjenom linearne regresije.

MT se koristi kao model pokreta za predviđanje učinkovitosti unosa teksta. Osim vremenskog modela pokreta, potreban je i jezični model (P) za ciljani. Jezični model sadrži frekvencije pojavljivanja digrama u odabranom korpusu. Sljedeći korak je izračunavanje prosječnog vremena pomaka (CT) između dvije tipke na zadanoj tipkovnici, koristeći dobivene modele pokreta i jezični model(P):

$$CT = \sum_{i \in C} \sum_{j \in C} (P_{ij} \cdot MT_{ij}) \tag{4.6}$$

CT predstavlja teoretski minimalno vrijeme, jer ne uključuje mentalne aktivnosti koje korisnik uobičajeno mora obaviti.

Iz dobivenog CT-a može se izračunati broj utipkanih znakova u sekundi (CPS_max) kao inverz CT-a:

$$CPS_{\max} = \frac{1}{CT} \quad [\mathrm{s}^{-1}] \tag{4.7}$$

Na kraju, pomoću dobivenog CPS_max, može se izračunati predviđena brzina utipkavanja (WPM_max) pomoću sljedećeg izraza:

$$WPM_{\max} = CPS_{\max} \cdot \frac{60}{5} \tag{4.8}$$

U izrazu za WPM, 60 predstavlja broj sekundi u minuti, dok 5 predstavlja broj znakova u riječi.

4.2 HCI eksperiment

U organiziranom eksperimentu sudjelovalo je dvanaest ispitanika. Njihove godine bile su u rasponu od 17 do 28. Svi sudionici bili su redoviti korisnici računala. Ispitanici nisu imali prethodno iskustvo s korištenjem bilo kojeg oblika tehnologije praćenja očiju ili virtualnih tipkovnica. Eksperiment je proveden u 2x2 dizajnu s ponavljanim mjerenjima. Postoje dvije nezavisne varijable: raspored tipkovnice (tipkovnica s kvadrantima, QWERTY) i modalitet unosa (pomak lijevo-desno, slobodna putanja pokazivača). Tijekom eksperimenta mjerene su dvije zavisne varijable: broj riječi po minuti (engl. *Words Per Minute, WPM*) i ukupna stopa pogreške (engl. *Total Error Rate*).

Jedan od problema koji se može javiti u eksperimentu s ponavljanim mjerenjima je efekt učenja (engl. *learning effect*). Kako ispitanici prolaze kroz testove, njihove performanse mogu se poboljšati zbog stjecanja dodatnog iskustva sa svakim odrađenim testom. To može dovesti do iskrivljenih rezultata eksperimenta. Utjecaj efekta učenja može se kompenzirati korištenjem metoda uravnoteženja. Metoda uravnoteženja provodi se tako da se sudionici raspoređuju u grupe, pri čemu se uvjeti predstavljaju svakoj grupi u različitim redoslijedima [16]. U ovom eksperimentu kao metoda uravnoteženja korišten je balansirani latinski kvadrat (Slika 4.1). U balansiranom latinskom kvadratu svaki uvjet pojavljuje se točno jednom u svakom retku i stupcu. Nadalje, svaki se uvjet pojavljuje prije i iza drugog uvjeta jednak broj puta. Na primjer, uvjet B slijedi uvjet A dva puta i također dva puta prethodi uvjetu A [16]. Važno je napomenuti da se balansirani latinski kvadrat može napraviti samo za parni broj uvjeta.

4.2.1 Aplikacije FittsStudy i TextTest

Aplikaciju FittsStudy razvili su Jacob O. Wobbrock, Susumu Harada, Edward Cutrell i I. Scott MacKenzie 2011. godine. FittsStudy služi za provođenje i analizu istraživanja vezanih za operaciju pokazivanja i odabira u skladu s utvrđenim međunarodnim i akademskim standardima [17]. Omogućena je podrška za 1D i 2D zadatke pokazivanja i odabira. Aplikacija podržava manipulaciju parametara A, W

A	B	D	C
B	C	A	D
C	D	B	A
D	A	C	B

Slika 4.1 *Balansirani latinski kvadrat 4x4*

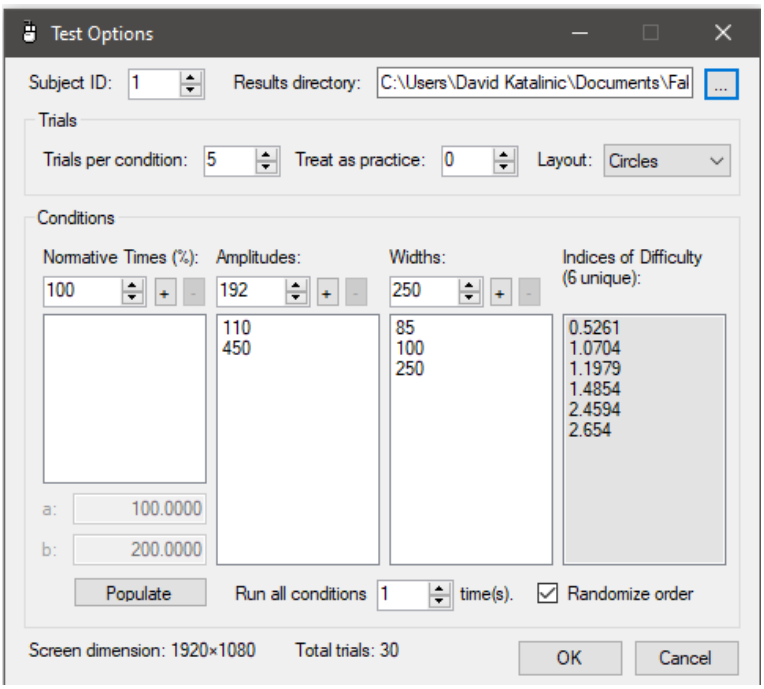
te manipulaciju MT s metronomom [17]. Aplikacija bilježi sve pokušaje vezane za operacije pokazivanja i odabira tijekom jednom sjednice u XML formatu, na temelju kojeg onda kasnije provodi statističku analizu. Ostale značajke uključuju automatski izračun modela i mjera, izvoz rezultata u proračunske tablice te alat za vizualno istraživanje obavljenih zadataka.

Pritiskom kombinacije tipki "Ctrl + N" u aplikaciji otvara se izbornik za postavljanje parametara eksperimenta. Izbornik se može vidjeti na slici 4.2.

U izborniku, glavni parametri koje se mogu postaviti su [19]:

- Identifikator ispitanika (engl. *Subject ID*) - Jedinstvena oznaka korisnika
- Raspored (engl. *Layout*) - Padajući izbornik u kojem se bira između opcija Ribbons (1D zadaci) i Circles (2D zadaci);
- Pokušaji po uvjetu (engl. *Trials per condition*) - Broj meta koji se pojavljuje po uvjetu;
- Tretirati kao vježbu (engl. *Treat as practice*) - Broj meta koje će se tretirati kao vježba;
- Amplituda - Udaljenosti meta koje se pojavljuju u zadacima;
- Širina - Širine meta koje se pojavljuju u zadacima.

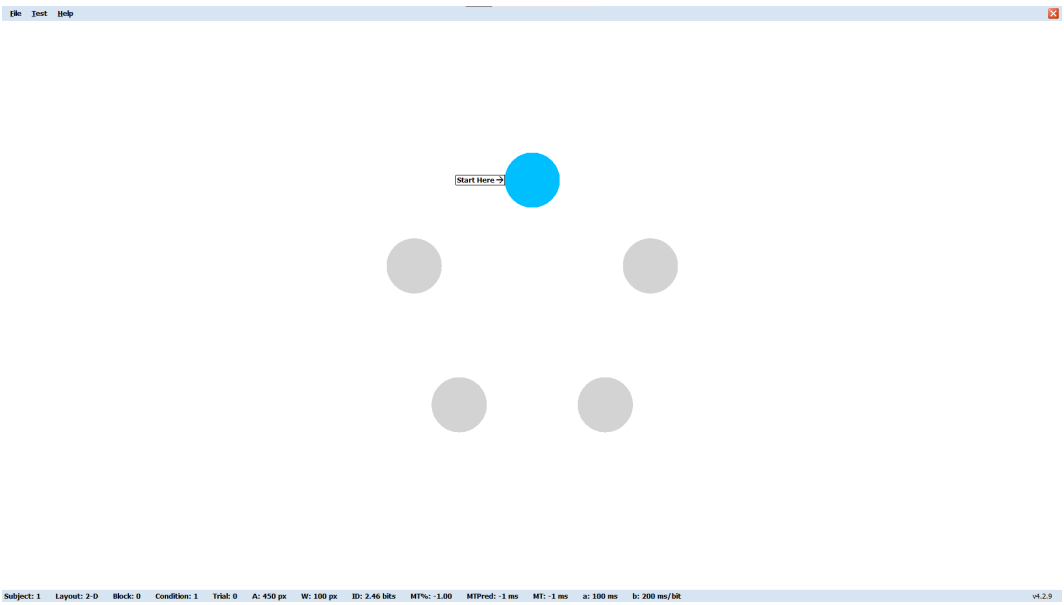
Za svaku seriju postavljeno je 5 meta (*Trials per condition*). Svaka meta za amplitudu ima dvije moguće vrijednosti: 110 i 450 dok svaka meta ima tri moguće širine: 85,



Slika 4.2 *Izbornik za postavljanje parametara u aplikaciji FittsStudy*

100 i 250. Za sve kombinacije amplitude i širine mete ukupno postoji 6 jedinstvenih indeksa složenosti zadataka. Broj meta koji će se tretirati kao vježba je postavljen na nulu. Za raspored (*Layout*) odabrane su kružnice. Na samom kraju, odabran je nasumični raspored kombinacija amplituda i širina.

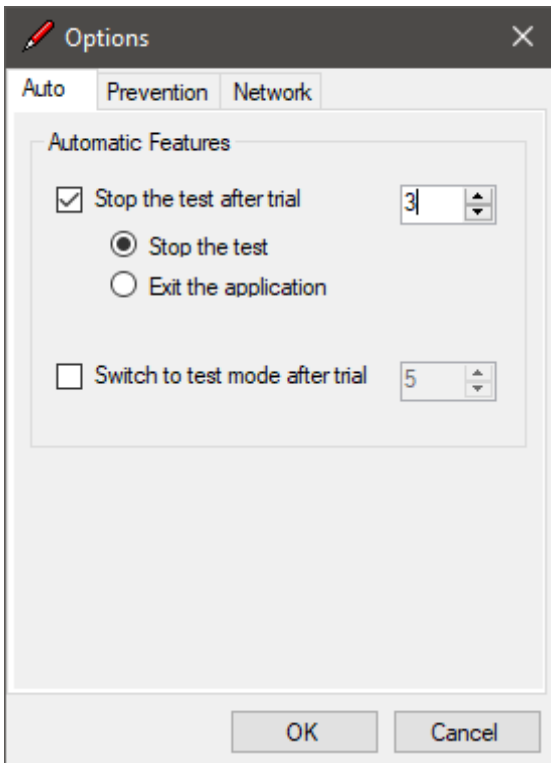
Pritiskom na gumb "OK" započinje prva serija testiranja. Na ekranu se pojavljuje broj meta jednak broju koji je definiran u *Trials per condition*. Početna meta označena je porukom "Start here", a mjerenje vremena počinje tek kada ispitanik klikne na tu početnu metu. Meta koju ispitanik mora kliknuti označena je plavom bojom. U slučaju da ispitanik promaši metu, ona će se zacrveniti, a sljedeća meta će se označiti plavom bojom. Kada korisnik odabere sve mete završava prva serija. U svakoj seriji mijenja se kombinacija amplitude i širine mete. Nakon završetka svih serija, rezultati se spremaju u xml datoteku.



Slika 4.3 *Primjer jedne testne sjednice u aplikaciji FittsStudy*

Aplikacija TextTest [18] služi za provođenje eksperimenata unosa teksta i analizu rezultata. Aplikaciju je razvio Jacob O. Wobbrock za Windows operacijski sustav. Kao i u FittsStudy aplikaciji, svi rezultati se spremaju u xml datoteke koje se onda kasnije koriste za analizu. Podržane su brojne metrike unosa teksta, mjere stope pogrešaka na razini znakova te matrice konfuzije na razini znakova. Aplikacija dolazi s već učitanim skupom fraza MacKenzie i Soukoreff (2003), ali korisnik može unijeti i vlastite fraze. Dodatno je podržana mogućnost slanja poruka drugim aplikacijama putem protokola TCP te prema web stranicama korištenjem web socket-a.

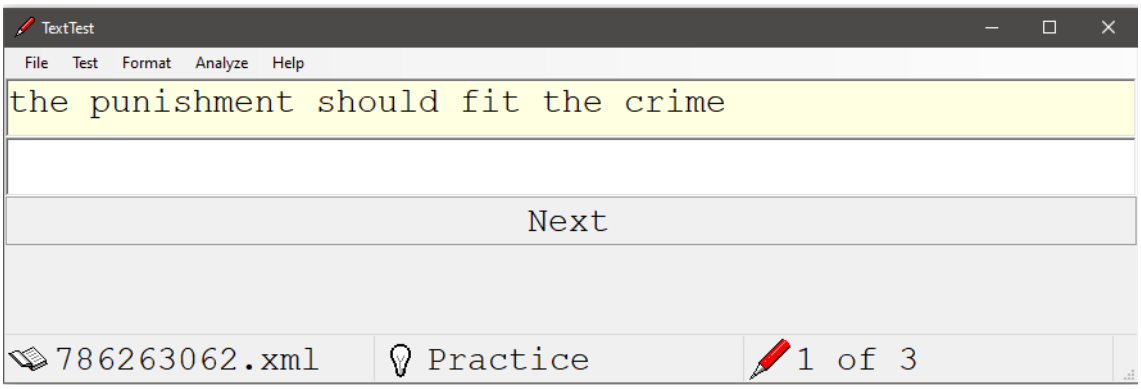
Nakon pokretanja aplikacije, klikom na "Test" u alatnoj traci i odabirom opcije "Options" iz padajućeg izbornika, otvara se izbornik za postavljanje opcija eksperimenta (Slika 4.4). U kartici "Auto" korisnik može odrediti nakon koje fraze će se testiranje zaustaviti te koji će se broj fraza tretirati kao vježba. U kartici "Prevention" moguće je odrediti znakove (interpunkcija, brojevi...) koji se ne smiju pojavljivati u frazama. Na kraju, kartica "Network" omogućuje definiranje veze preko koje će se slati i primati poruke od drugih aplikacija.



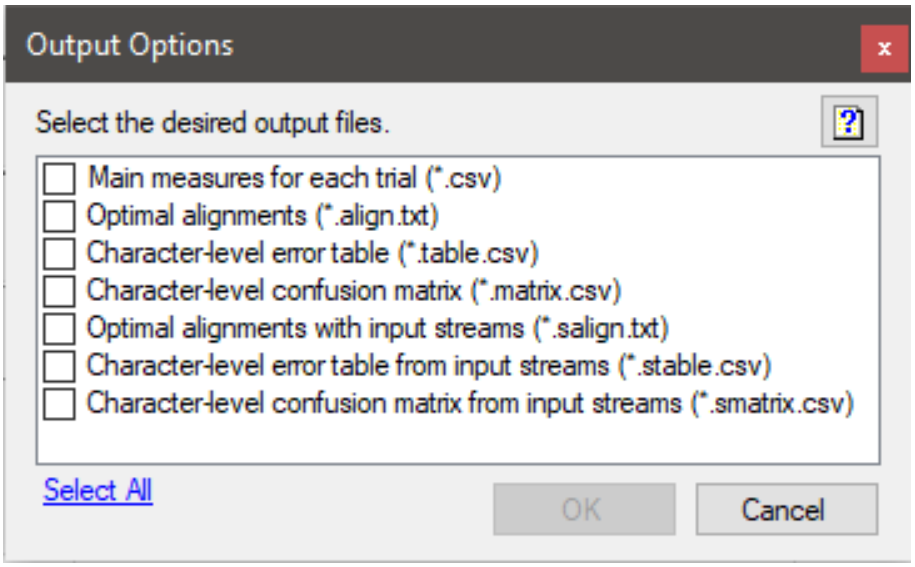
Slika 4.4 *Izbornik za postavljanje opcija eksperimenta u aplikaciji TextTest*

Kombinacijom tipki "Ctrl + N" pokreće se novi test nakon što su sve postavke eksperimenta postavljene (Slika 4.5). U prvom polju prikazana je fraza koju ispitanik mora unijeti u drugo polje. Kada je fraza u potpunosti unesena pritiskom na gumb "Next" prelazi se na sljedeću frazu. U slučaju da nije postavljeno nakon koje fraze će se testiranje zaustaviti, test se može završiti pritiskom tipke "Esc". Po završetku testiranja, rezultat se sprema u XML datoteku, a ispitaniku se prikazuje novi prozor s grafovima za WPM i Error Rate.

Po završetku testiranja, klikom na "Logs" u kartici "Analyze" i odabirom generirane XML datoteke, otvara se izbornik "Output Options" (Slika 4.6). U ovom izborniku korisnik može odabrati koje podatke želi da izlazna datoteka sadržava. Neki od opcija koje može odabrati su: glavni rezultati (WPM, TER), mjera stope pogreške na razini znaka, matrica konfuzije na razini znakova, optimalna poravnanja i sl.



Slika 4.5 *Primjer jediničnog zadatka u aplikaciji TextTest*



Slika 4.6 *Odabr izlaznih datoteka u TextTest aplikaciji*

4.2.2 Upitnici

Po završetku svih testiranja, ispitanik dobiva NASA-TLX (engl. *NASA Task Load Index*) [20] upitnik. NASA-TLX je subjektivni, višedimenzionalni alat za procjenu koji ocjenjuje percipirano opterećenje prilikom izvođenja nekog zadatka. Razvijen je od strane Human Performance Group u NASA-inom Ames Research centru 1988.

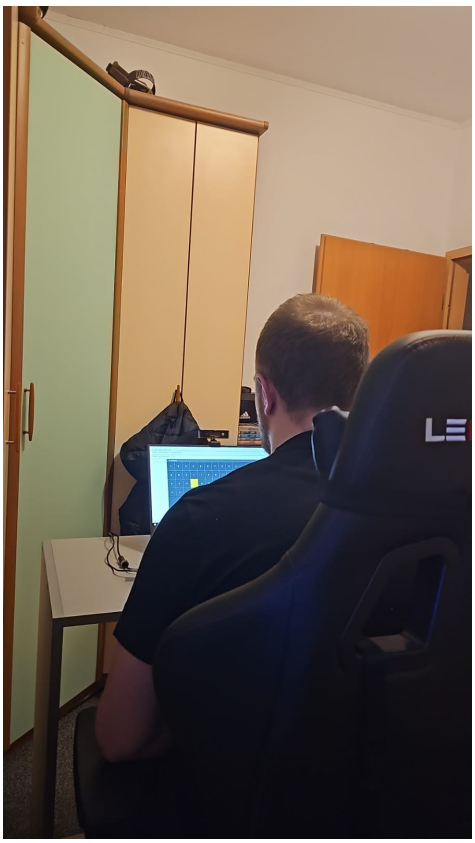
godine te je danas citiran preko 15000 puta [21]. Upitnik procjenjuje percipirano opterećenje pomoću 5 faktora koji se boduju na skali od 1 do 21; temeljeno na odgovarajućim pitanjima kako slijedi:

- **Mentalni zahtjevnost** – Koliko je mentalne i perceptivne aktivnosti bilo potrebno? Je li zadatak bio jednostavan ili zahtjevan, jednostavan ili složen, precizan ili kompleksan?
- **Fizički zahtjevnost** – Koliko je fizičke aktivnosti bilo potrebno? Je li zadatak bio lagan ili zahtjevan? Spor ili brz? Opušten ili naporan? Odmarajući ili mukotrpan?
- **Razina frustracije** – Koliko ste se osjećali nesigurno, obeshrabreno, iritirano, pod stresom i nervozno, naspram sigurnosti, zadovoljstva, opuštenosti i samozadovoljstva tijekom zadatka?
- **Uloženi napor** – Koliko ste se morali truditi (mentalno i fizički) da biste postigli svoju razinu izvedbe?
- **Izvedba** – Koliko mislite da ste bili uspješni u postizanju ciljeva zadatka koje je postavio eksperimentator (ili vi sami)? Kakva je vaša razina zadovoljstva?

4.2.3 Procedura eksperimenta

Eksperiment je proveden u tihom prostoru u kojem su sudionici prilagodili svoje pozicije na stolici s podesivom visinom. Nakon toga, kamera se prilagodila za specifičnog sudionika (pozicija, kut, osvjetljenje). Nije bilo uvodnog treninga, sudionik je odmah započeo s prvim testom. Istraživač je objasnio svaki zadatak u eksperimentu. Nadalje, sudionik je dobio kratku prezentaciju dviju metoda unosa teksta pomoću pomaka očiju i neke smjernice o tome kako koristiti svaku tipkovnicu. Prvo je provedeno eksperimentalno modeliranje pomoću aplikacije FittsStudy. U aplikaciji FittsStudy postavljeni su parametri testa kao što je prikazano na slici 4.2. Nakon završetka prediktivnog modeliranja provedeno je empirijsko vrednovanje koristeći aplikaciju TextTest. Sudionici su trebali unijeti 3 rečenice, od kojih svaka sadrži od 3 do 5 riječi. Rečenice su nasumično birane iz skupa od 40 rečenica. Ispravljanje riječi bilo je opcionalno. Ukupno je izvedeno 4 testa po sudioniku. Prvi test uklju-

čivao je unos teksta koristeći tipkovnicu s kvadrantima i metodu unosa lijevo-desno, dok je drugi test koristio tipkovnicu s kvadrantima i metodu slobodna putanja pokazivača. Treći i četvrti test koristili su QWERTY tipkovnicu, ali je treći test koristio je metodu unosa lijevo-desno (Slika 4.7), dok je četvrti test koristio metodu slobodne putanje pokazivača. Sudionici su imali mogućnost odmora između svakog testa. Za minimalizaciju efekta učenja korišten je balansirani latinski kvadrat 4x4 (Slika 4.1). Prosječno trajanje eksperimenta bilo je oko 2 sata i 15 minuta za svih 4 testa. Nakon završetka svih testova, sudionici su zamoljeni da popune upitnik temeljen na NASA-TLX-u, koji je kasnije analiziran koristeći samo "sirove" TLX odgovore.



Slika 4.7 *Sudionik koji izvodi test koristeći QWERTY tipkovnicu i modalitet pomak lijevo-desno*

Poglavlje 5

Analiza rezultata

U ovom poglavlju prikazani su rezultati empirijskog vrednovanja (WPM, ukupnu stopu pogrešaka i percipirano radno opterećenje tijekom interakcije) te prediktivnog modeliranja. Za analizu podataka dobivenih iz aplikacije TextTest korištena je parametarska dvofaktorska ANOVA s ponovljenim mjerenjima (engl *Two-Way Repeated Measures ANOVA, RM ANOVA*) iz Python knjižnice *stats* koja je dio *statsmodels* paketa. Za podatke dobivene iz NASA-TLX upitnika korišten je Wilcoxonov test (engl *Wilcoxon signed rank test*) u SPSS alatu.

5.1 Analiza rezultata eksperimenta

Za analizu rezultata eksperimenta korištena je dvofaktorska ANOVA s ponavljanim mjerenjima. Ova statistička metoda omogućava ispitivanje utjecaja dvaju nezavisnih varijabli (faktora) na zavisne varijable. U ovom eksperimentu zavisne varijable su WPM i TER. Dizajn s ponavljajućim mjerenjima označava da su svi ispitanici odradili sve testove pod istim uvjetima. Za korištenje dvofaktorske ANOVA-e podaci moraju biti normalno distribuirani te mora biti zadovoljen uvjet sferičnosti (engl *sphericity*). Sferičnost je pretpostavka da su varijance razlika između svih kombinacija razina nezavisnih varijabli jednake. Za provjeru uvjeta sferičnosti koristi se Mauchlyjev test sferičnosti (engl *Mauchly's sphericity test*). Ako Mauchlyjev test pokaže kršenje sferičnosti, moguće je koristiti korekcije kao što su Greenhouse-Geisser ili

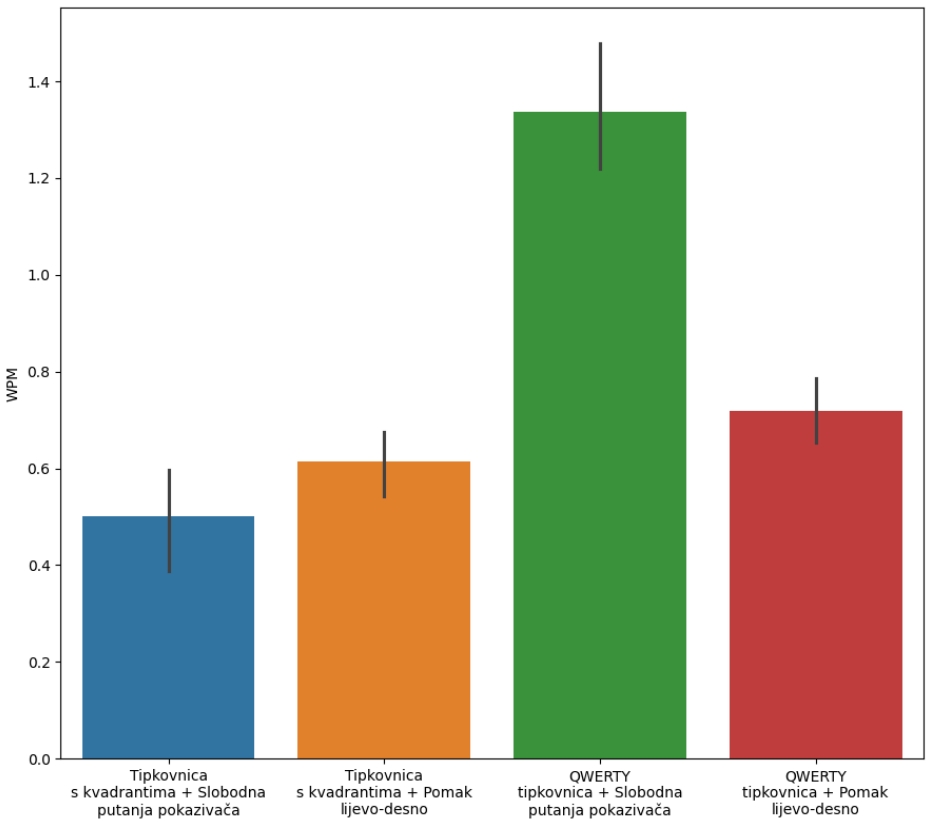
Huynh-Feldt.

5.1.1 Brzina unosa teksta

Rezultati za WPM prikazani su na slici 5.1. Analiza srednjih vrijednosti otkriva da je prosječni WPM za kombinaciju tipkovnice s kvadrantima i slobodne putanje pokazivača bio 0,50. Nasuprot tome, tipkovnica s kvadrantima s modalitetom pomaka lijevo-desno imala je prosječni WPM od 0,61, što predstavlja povećanje brzine unosa od 22%. Dalje, QWERTY tipkovnica s pomakom lijevo-desno pokazala je dodatno poboljšanje, s prosječnim WPM-om od 0,72, što znači povećanje brzine od 44% u odnosu na prvu kombinaciju. Na kraju, kombinacija QWERTY tipkovnice i slobodne putanje pokazivača bila je 268% brža, s WPM vrijednošću od 1,34.

Na slici 5.2 ilustrirana je grafička usporedba minimalne, srednje i maksimalne vrijednosti WPM-a po unesenim frazama. Za sve tri vrijednosti, WPM izrazito poraste nakon prve unesene fraze nakon čega se vrijednost stabilizira. Najveći porast između prve i druge fraze dogodi se za minimalnu vrijednost WPM-a.

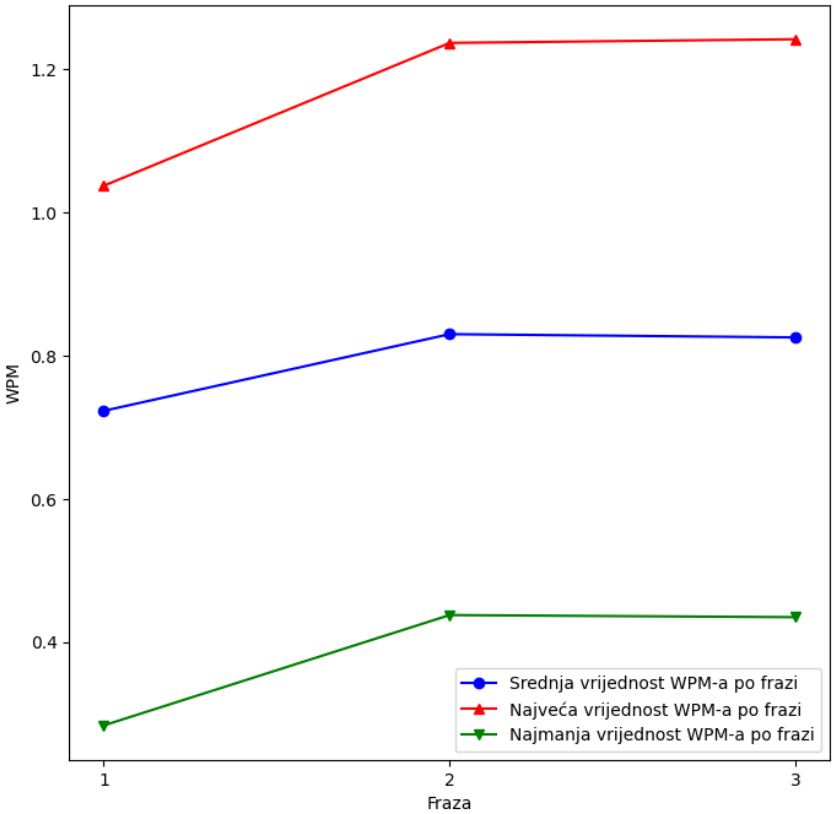
Korištenjem dvofaktorska ANOVA-e s ponavljanim mjerenjima, otkriven je značajan utjecaj rasporeda tipkovnice na WPM: $F(1, 11) = 74{,}0132$, $p < .0001$. Slično tome, metoda unosa je također imala statistički značajan utjecaj na WPM: $F(1,11) = 21{,}6702$, $p < 0{,}001$. Uz to, analiza je otkrila značajnu interakciju između rasporeda tipkovnice i metode unosa: $F(1,11) = 52{,}1988$, $p < 0{,}0001$. QWERTY tipkovnica pokazala se značajno bržom (MEAN =1.02788, STD = 0.402406) od tipkovnice s kvadrantima (MEAN = 0.557952, STD = 0.202863). Ova razlika u brzini može se pripisati većem iskustvu sudionika u radu s QWERTY tipkovnicom. S obzirom na to da su sudionici unijeli samo 6 fraza koristeći tipkovnicu s kvadrantima, nisu imali dovoljno vremena prilagoditi se novom rasporedu. Što se tiče metoda unosa, slobodna putanja pokazivača se pokazala značajno bržom (MEAN = 0.919094, STD = 0.498651) od pomaka lijevo-desno (MEAN = 0.666746 , STD = 0.184710), što je u skladu s prirodom dviju metoda. Pomak lijevo-desno omogućava pomicanje samo za jedno slovo lijevo ili desno, dok slobodna putanja pokazivača omogućava izravan prijelaz na željeno slovo.



Slika 5.1 *Srednja vrijednost WPM-a za sve kombinacije dizajna tipkovnice i modaliteta unosa*

5.1.2 Ukupna stopa pogreške

Rezultati za ukupnu stopu pogreške prikazani su na slici 5.3. Analiza srednjih vrijednosti jasno pokazuje da kombinacije neovisnih varijabli koje koriste slobodnu putanju pokazivača imaju znatno višu stopu pogreške u usporedbi s kombinacijama koje koriste metodu unosa pomak lijevo-desno. Prosječna stopa pogreške za QWERTY tipkovnicu s metodom slobodne putanje pokazivača iznosi 0,27. Tipkovnica s kvadrantima u kombinaciji s istom metodom unosa pokazuje prosječnu stopu pogreške od 0,21, što je smanjenje od 22,23% u odnosu na prvu kombinaciju. Nadalje, tipkovnica s kvadrantima u kombinaciji s metodom unosa pomak lijevo-desno ima prosječnu stopu pogreške od 0,14, što predstavlja smanjenje od 48,14%. Na kraju, kombinacija



Slika 5.2 *Vrijednost WPM-a po unesenim frazama*

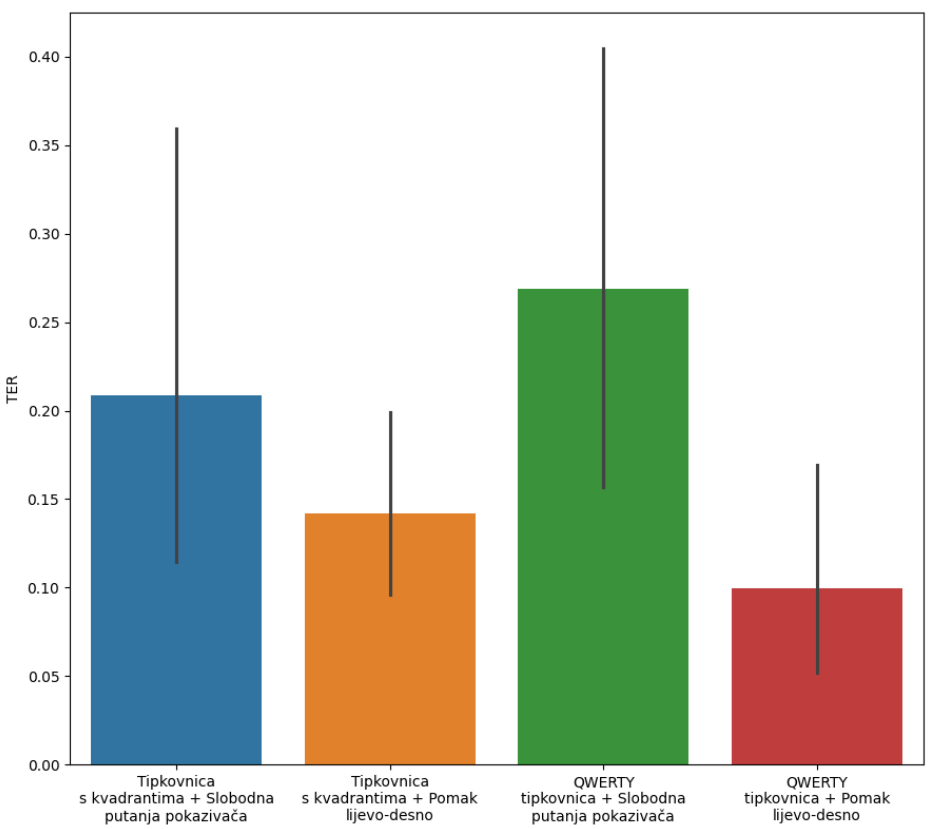
QWERTY tipkovnice i metode unosa pomak lijevo-desno postiže najnižu prosječnu stopu pogreške od 0,10, što je smanjenje od 62,97% u odnosu na početnu kombinaciju.

Grafička usporedba minimalne, srednje i maksimalne vrijednosti TER-a po unesenim frazama prikazana je na slici 5.4. Maksimalna vrijednost se naglo smanji u drugi frazi, ali zato opet poraste u trećoj frazi. Srednja vrijednost ima izrazito mali pad u drugoj frazi, ali mimo toga ostaje relativno konzistentna kroz sve fraze. S druge strane, minimalna vrijednost postepeno raste kroz unesene fraze.

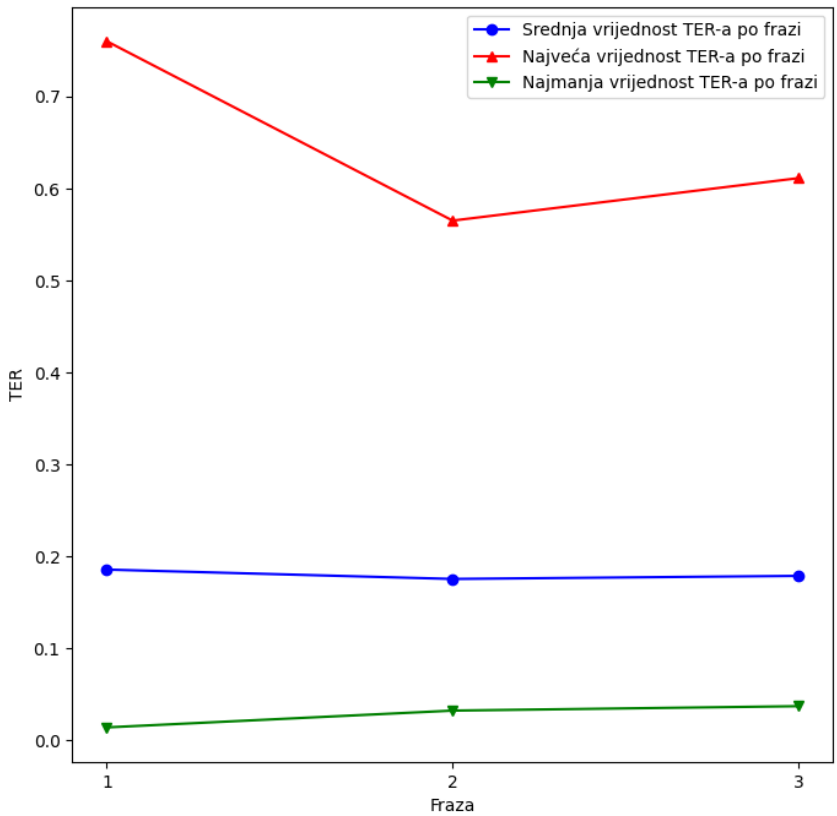
Isti statistički test koji je korišten za analizu WPM-a primijenjen je i za analizu ukupne stope pogreške. Rezultati su pokazali značajan utjecaj metode unosa teksta na ukupnu stopu pogreške ($F(1, 11) = 5{,}9538$, $p < 0{,}05$), dok raspored tipkovnice

(F(1, 11) = 0.078, p = 0,7951) i interakcija između rasporeda i metode unosa (F(1, 11) = 3,4182, p = 0,0915) nisu imali statistički značajan utjecaj.

Značajno je da je stopa pogreške za slobodnu putanju pokazivača (MEAN = 0.238759, STD = 0.233727) bila značajno veća u usporedbi s pomakom lijevo-desno (MEAN = 0.120825, STD = 0.139070). Ova razlika može se pripisati postavci za treptanje, koje simulira klik, za obje metode. Kod pomaka lijevo-desno, prag za treptanje bio je postavljen na 1,5 sekundi, dok je za metodu slobodna putanja pokazivača prag smanjen na 0,5 sekundi, kako bi se omogućilo brže iniciranje klika zbog stalnog kretanja pokazivača. Ta razlika nije problematična u pomaku lijevo-desno, gdje se označena tipka može promijeniti samo kada korisnik usmjeri pogled lijevo ili desno.



Slika 5.3 *Stopa pogreške za sve kombinacije dizajna tipkovnice i modaliteta unosa*



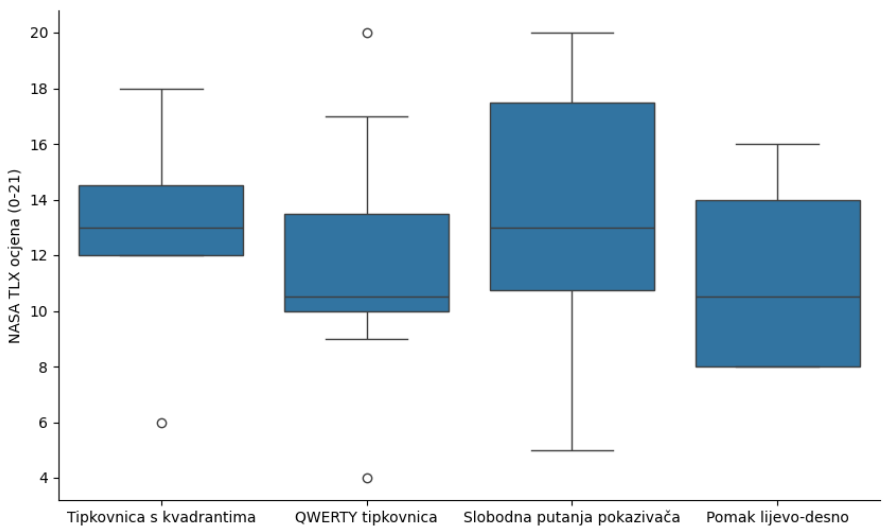
Slika 5.4 *Vrijednost TER-a po unesenim frazama*

5.2 Analiza rezultata upitnika

Svim ispitanicima nakon obavljenog eksperimenta dan je NASA-TLX upitnik. Za analizu podataka dobivenih iz NASA-TLX upitnika korišten je Wilcoxonov test. Wilcoxon test je neparametarski test koji se koristi za analizu rezultata dobivenih iz ponavljanih mjerenja. Ovaj test se može koristiti samo ako se uspoređuju dva uzorka. U slučaju da se uspoređuju 3 ili više uzorka potrebno je koristiti Friedmanov test. Za razliku od eksperimentalnih rezultata, kod analize podataka iz upitnika svaka se nezavisna varijabla analizira zasebno.

5.2.1 Mentalna zahtjevnost

Slika 5.5 prikazuje rezultate procjene mentalne zahtjevnosti za različite razine nezavisnih varijabli. Iz grafa je vidljivo da tipkovnice s kvadrantima uzrokuje nešto veću mentalnu zahtjevnost u usporedbi s QWERTY tipkovnicom. Što se tiče modaliteta interakcije, slobodno kretanje pokazivača pokazalo se mentalno zahtjevnijim.



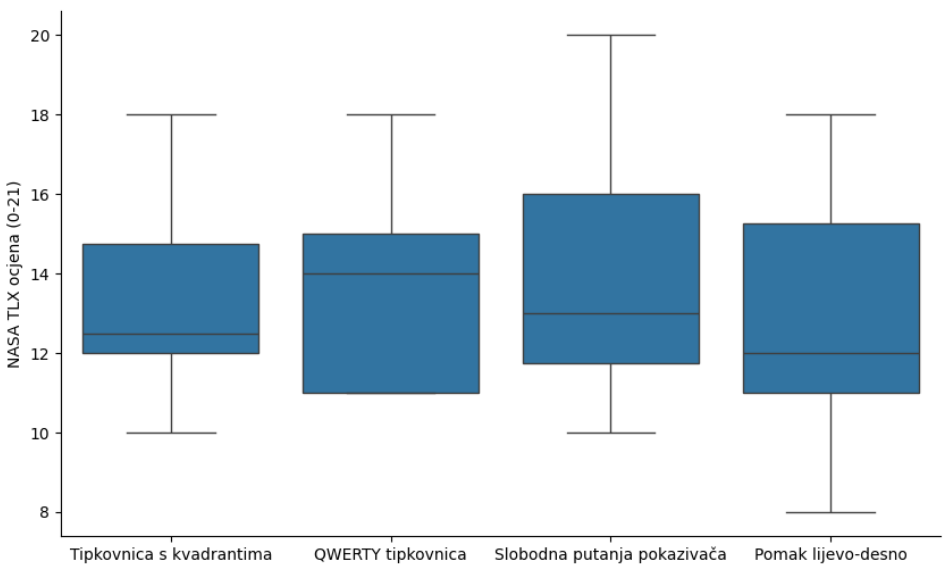
Slika 5.5 *Mentalna zahtjevnost*

Korištenjem Wilcoxonovog testa dobiveni su sljedeći rezultati:

- Nije pronađena značajna razlika u mentalnom opterećenju između QWERTY tipkovnice i tipkovnice s kvadrantima; Z = -1.753, p = 0.080.
- Mentalno opterećenje bilo je značajno niže pri korištenju pomaka lijevo-desno u usporedbi s metodom slobodne putanje pokazivača; Z = -2.139, p = 0.032.

5.2.2 Fizička zahtjevnost

Na grafu prikazanom na slici 5.6 ilustrirana je grafička usporedba ocjene fizičke zahtjevnosti za različite razine nezavisnih varijabli. Iz grafa je vidljivo da su rezultati za tipkovnicu s kvadrantima i QWERTY tipkovnicu izrazito slični. Kod modaliteta interakcije, isto ne postoji značajna razlika u percipiranoj fizičkoj zahtjevnosti.



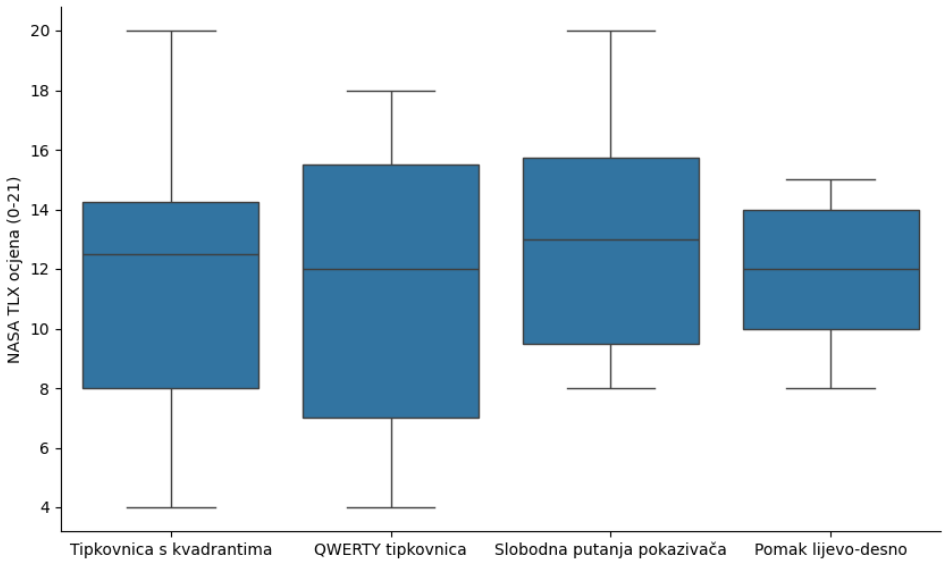
Slika 5.6 *Fizička zahtjevnost*

Rezultati analize pokazuju:

- Nije pronađena značajna razlika u fizičkom opterećenju između QWERTY tipkovnice i tipkovnice s kvadrantima; Z = 0.000, p = 1.000.
- Nije pronađena značajna razlika u fizičkom opterećenju između metode slobodna putanja pokazivača i metode pomak lijevo-desno; Z = -0.938, p = 0.348.

5.2.3 Frustracija

Grafički prikaz ocjena razine frustracije za različite razine nezavisnih varijabli prikazan je na slici 5.7. Iz grafa je očito da ne postoji značajna razlika u percipiranoj frustraciji, bilo između različitih dizajna tipkovnica, bilo između različitih metoda unosa.



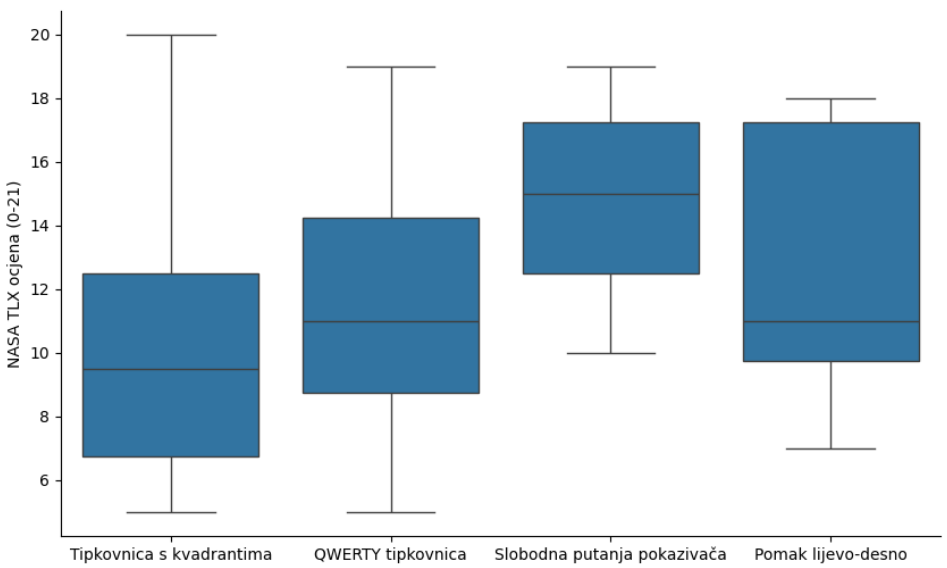
Slika 5.7 *Frustracija*

Korištenjem Wilcoxonovog testa dobiveni su sljedeći rezultati:

- Nije pronađena značajna razlika u frustraciji između QWERTY tipkovnice i tipkovnice s kvadrantima; Z = -0.059, p = 0.953.
- Nije pronađena značajna razlika u frustraciji između metode slobodna putanja pokazivača i metode pomak lijevo-desno; Z = -1.283, p = 0.199.

5.2.4 Izvedba

Na slici 5.8 dan je grafički prikaz procijenjene razine izvedbe po razinama nezavisnih varijabli. Iz grafa je vidljivo da tipkovnica s kvadrantima bilježi slabije performanse u odnosu na QWERTY tipkovnicu. Slobodna putanja pokazivača pokazuje manju varijabilnost od pomaka lijevo-desno te ima veći medijan.



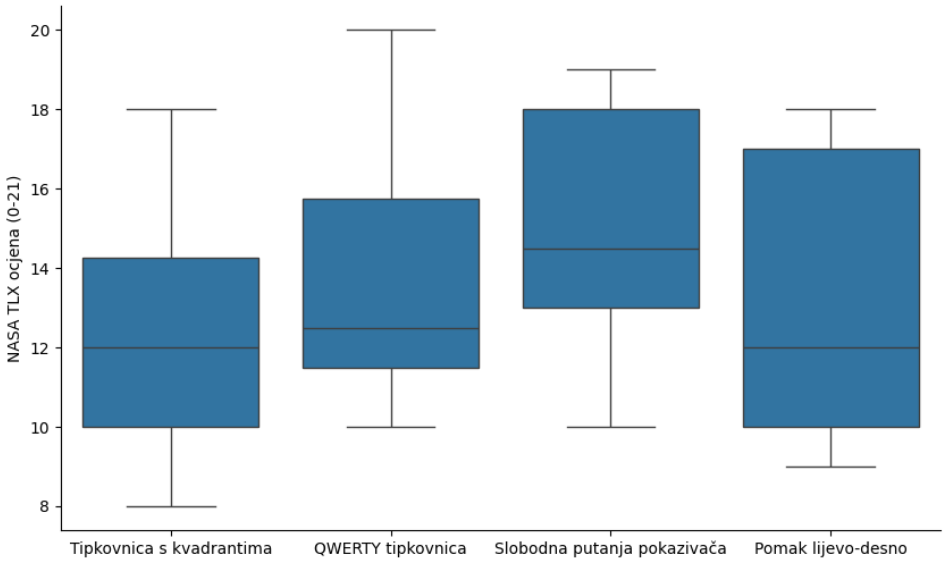
Slika 5.8 *Izvedba*

Analizom su dobiveni sljedeći rezultati:

- Procijenjena razina izvedbe je bila značajno viša pri korištenju QWERTY tipkovnice u usporedbi s tipkovnicom s kvadrantima; Z = -2.496, p = 0.013.
- Nije pronađena značajna razlika u percipiranoj izvedbi između metode slobodna putanja pokazivača i metode pomak lijevo-desno; Z = -1,797, p = 0,072.

5.2.5 Trud

Grafička usporedba ukupno uloženog truda po razinama nezavisnih varijabli ilustrirana je na slici 5.9.Nema značajne razlike u percipiranom trudu između dvaju dizajna tipkovnica. Međutim, slobodna putanja pokazivača zahtijeva značajno veći trud kod modaliteta interakcije, što je rezultat potrebe za kontinuiranim usmjeravanjem pokazivača i brzim treptanjem kako bi se simulirao klik.



Slika 5.9 *Trud*

Analizom su dobiveni sljedeći rezultati:

- Nije pronađena značajna razlika u ukupno uloženom naporu između QWERTY tipkovnice i kvadrantne tipkovnice; Z = -1.128, p = 0.259.
- Percipirani trud bio je značajno viši pri korištenju metode slobodna putanja pokazivača u usporedbi s metodom pomak lijevo-desno; Z = -2.368, p = 0.018

5.3 Prediktivni modeli

U ovom potpoglavlju predstavljeni su rezultati prediktivnog modeliranja. Za svaku kombinaciju razina nezavisnih varijabli izračunata je predviđena gornja granica brzine unosa teksta. Gornja granica brzine određena je s pomoću jednadžbi 4.6, 4.7, 4.8. Prije korištenja ovih jednadžbi bilo je potrebno izračunati MT.

Za modalitet interakcije slobodna putanja pokazivača, korištena je aplikacija FittsStudy (Slika 4.3) koja kao rezultat daje MT izražen s pomoću jednadžbe 4.5. U dobivenoj jednadžbi za MT parametar a iznosi 818.5362 dok parametar b iznosi 6605.2352. Pomoću dobivene jednadžbe potrebno je izračunati MT za svaku kombinaciju digrama. Za izračun MT-a, prvo je potrebno odrediti amplitude za svaki digram i širinu svake tipke jer se ti parametri koriste u izračunu indeksa složenosti. Razvijena je jednostavna skripta koja izračunava amplitude i širine za sve moguće digrame za obje vrste tipkovnica, s obzirom da su obje tipkovnice izvorno dizajnirane te imaju poznatu geometriju. Nadalje, implementirana je i skripta koja računa MT za sve moguće digrame s pomoću dobivenih amplituda i širina. Nakon što je MT izračunat, mogu se izračunati CT i CPSmax, a na kraju i WPMmax. U svrhu toga, implementirana je skripta koja kao parametre prima datoteku s dobivenim MT-om i jezični model P s pomoću kojih onda računa WPMmax. Za jezični model P korišten je korpus hrenWaC koji se sastoji od 27 znakova (engleska abeceda + "Space"). Na slici 5.10 može se vidjeti struktura jezičnog modela.

Za pomak lijevo-desno nije moguće koristiti aplikaciju FittsStudy jer ovaj modalitet interakcije nije klasična akcija pokazivanja i odabira. Iz tog razloga, za potrebe izračuna MT-a razvijene su dvije skripte koje računaju optimalni MT za svaki par digrama, za svaku od tipkovnica. U skriptama se koriste vremenske konstante za kretanje lijevo i desno i simulaciju klika. Vremenske konstante predstavljaju vremenski prag koliko ispitanik mora gledati u jednu stranu ili treptati da se izvrši odgovarajuća akcija. Implementirane funkcije izračunavaju vrijeme potrebno za prelazak s jedne tipke na drugu, pritom birajući najkraći put između njih. Za tipkovnicu s kvadrantima dodatno je uzeto u obzir vrijemo potrebne da se prijeđe u odgovarajući kvadrant. Važno je napomenuti da dobiveno vrijeme predstavlja teoretski najkraće moguće, budući da ne uzima u obzir mentalne procese koje ispitanik mora obaviti.

```
a       0,005615635
aa      0,000052173
ab      0,001437081
ac      0,003420867
ad      0,002131932
ae      0,000092895
af      0,000541553
ag      0,001624803
ah      0,000041867
ai      0,001852030
aj      0,000108139
ak      0,000743016
al      0,008234013
am      0,001756416
an      0,014990797
ao      0,000026981
ap      0,001246138
aq      0,000018536
ar      0,008139973
as      0,005369442
at      0,010917661
au      0,000826464
av      0,001231681
aw      0,000270455
ax      0,000130182
ay      0,001179365
az      0.000098764
```

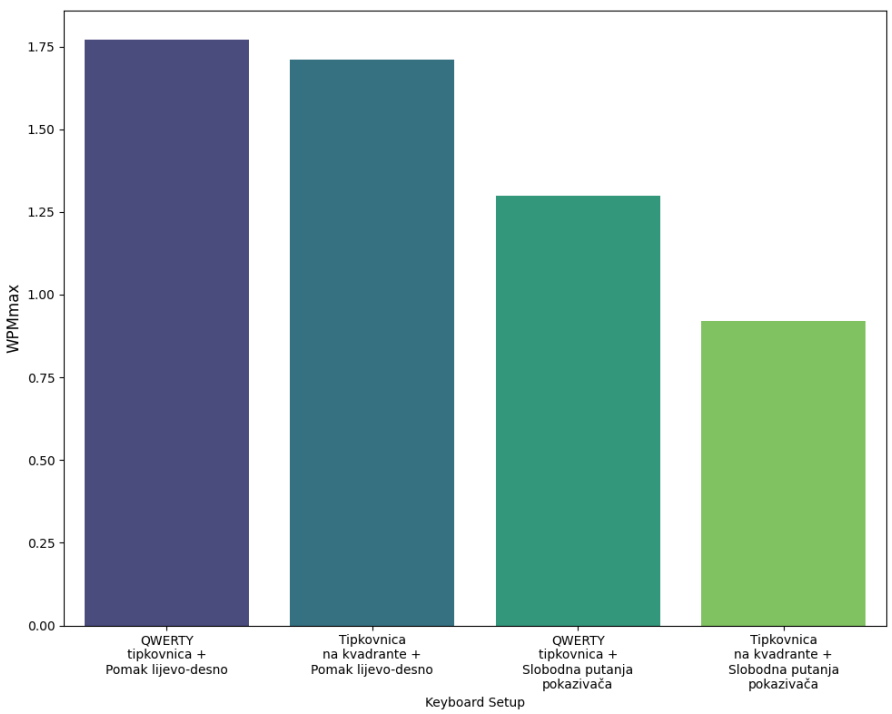
Slika 5.10 *Struktura jezičnog modela P*

Iz tog razloga, malo je vjerojatno da se ispitanik približi tom stupnju učinkovitosti. Na kraju, korištena je ista skripta za izračun WPMmax-a kao i za slobodnu putanju pokazivača.

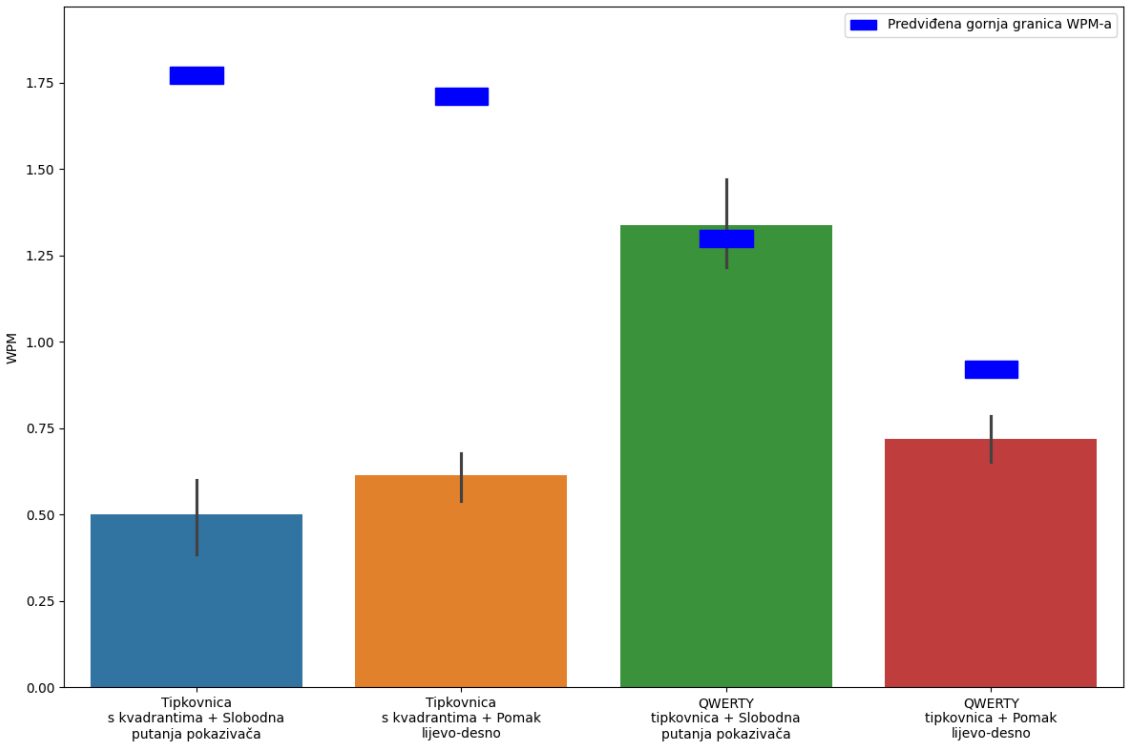
Grafička usporedba predviđenih gornjih granica brzine unosa teksta prikazana je na slici 5.11. Iz grafa je vidljivo da najveću predviđenu brzinu ima kombinacija QWERTY tipkovnice i pomaka lijevo-desno s predviđenim WPM-om od 1.77. Iza nje, slijedi kombinacija tipkovnice s kvadrantima i pomaka lijevo-desno s nešto nižim WPM-om od 1.71. Nakon toga, dolazi kombinacija QWERTY tipkovnice i slobodne putanje pokazivača koja ima WPM od 1.30. Na samom kraju, nalazi se kombinacija tipkovnice s kvadrantima i slobodne putanje pokazivača s WPM-om od 0.92.

Usporedbom rezultata prediktivnog modeliranja i empirijskog istraživanja (Slika

5.12) može se primijetiti da kombinacije koje uključuju pomak lijevo-desno kao modalitet interakcije imaju znatno niži stvarni WPM u usporedbi s predviđenim vrijednostima. Za QWERTY tipkovnicu s pomakom lijevo-desno, predviđeni WPM je 1.77 dok eksperimentalno dobiveni WPM iznosi 0.72. Slična situacija je i za kombinaciju tipkovnice s kvadrantima i pomaka lijevo-desno gdje prediktivni WPM iznosi 1.71, a stvarni WPM je 0.61. Ovo je očekivano jer prediktivno modeliranje daje teorijski najveće moguće WPM vrijednosti za pomak lijevo-desno. Nasuprot tome, predviđeni WPM za tipkovnicu QWERTY sa slobodnom putanjom pokazivača je manji od stvarnog. Predviđeni WPM iznosi 1.30 dok je stvarni WPM 1.34. Ovo se može objasniti nešto boljom izvedbom ispitanika tijekom eksperimenta u odnosu na prediktivno modeliranje. Za kombinaciju tipkovnice s kvadrantima i slobodne putanje pokazivača predviđeni WPM iznosi 0.92, a stvarni WPM iznosi 0.50.



Slika 5.11 *Predviđena gornja granica brzine unosa teksta*



Slika 5.12 *Usporedba rezultata prediktivnog modeliranja i empirijskog istraživanja*

Poglavlje 6

Zaključak

U ovom radu predstavljeno je istraživanje unosa teksta temeljenog na praćenju pokreta očiju na stolnim računalima s ciljem razumijevanja izvedivosti i izazova povezanih s ovom metodom interakcije. Empirijsko istraživanje je obuhvatilo testiranje dva rasporeda tipkovnica (kvadrantni i QWERTY) te dvije metode unosa ("lijevo-desno" i "slobodna putanja pokazivača"). Eksperimentalno vrednovanje je provedeno kroz niz testova unosa teksta koristeći TextTest aplikaciju. Metrike koje su se vrednovale su brzina unosa, ukupna stopa pogrešaka i percipirano opterećenje interakcije. Dodatno je provedeno prediktivno vrednovanje s ciljem predviđanja gornjih granica brzine unosa teksta.

Analiza brzine unosa teksta otkrila je da su korisnici značajno brži s QWERTY tipkovnicom. Ovo je rezultat manjeg iskustva ispitanika u radu s tipkovnicom s kvadrantima. Kod modaliteta interakcije, slobodna putanja pokazivača se pokazala značajno bržom od pomaka lijevo-desno. Ovo je očekivan rezultat s obzirom na prirodniji način funkcioniranja modaliteta slobodne putanje.

Analizom ukupne stope pogreške, utvrđeno je da slobodna putanja pokazivača ima značajno višu ukupnu stopu pogreške. To je rezultat implementacije treptanja, jer modalitet slobodna putanja pokazivača ima znatno niži vremenski prag za treptanje.

Rezultati NASA-TLX upitnika pokazali su da ispitanici smatraju slobodnu putanju pokazivača mentalno zahtjevnijom u usporedbi s pomakom lijevo-desno. Za

tipkovnice nije bilo razlike u percipiranoj mentalnoj zahtjevnosti. Ispitanici su sve tipkovnice i modalitet interakcije smatrali jednako fizički zahtjevnim te frustrirajućim. Percipirana razina izvedbe je bila značajno viša pri korištenju QWERTY tipkovnice dok za modalitete interakcije nije pronađena razlika u percipiranoj izvedbi. Naposljetku, ukupno uloženi trud je bio značajno viši pri korištenju modaliteta slobodna putanja pokazivača. Za tipkovnice nije pronađena značajna razlika u ukupnp uloženom trudu.

Bibliografija

[1] Majaranta, Päivi, Räihä, Kari-Jouko. (2007). 9. "Text Entry by Gaze: Utilizing Eye-Tracking. Text Entry Systems: Mobility, Accessibility, Universality",

[2] MacKenzie, I Ashtiani, Behrooz. (2011). BlinkWrite: Efficient text entry using eye blinks. Universal Access in the Information Society.

[3] MacKenzie, I.. (2011). Evaluating Eye Tracking Systems for Computer Input. Gaze Interaction and Applications of Eye Tracking: Advances in Assistive Technologies.

[4] Python. , s Interneta, `https://www.python.org/` , kolovoz 2024

[5] Python GazeTracking knjižnica, s Interneta, `https://github.com/antoinelame/GazeTracking`, kolovoz 2024.

[6] Python PyAutoGUI knjižnica, s Interneta, `https://pyautogui.readthedocs.io/en/latest/`, kolovoz 2024.

[7] Python tkinter knjižnica, s Interneta, `https://docs.python.org/3/library/tkinter.html` , kolovoz 2024.

[8] Python keyboard knjižnica, s Interneta, `https://pypi.org/project/keyboard/` , kolovoz 2024.

[9] Python OpenCV knjižnica, s Interneta, `https://docs.opencv.org/4.x/index.html`, kolovoz 2024.

[10] Python pynput knjižnica, s Interneta, `https://pynput.readthedocs.io/en/latest/`, kolovoz 2024.

[11] Python time knjižnica, s Interneta, `https://docs.python.org/3/library/time.html`, kolovoz 2024.

[12] Python threading knjižnica. s Interneta, `https://docs.python.org/3/library/threading.html`, kolovoz 2024.

[13] Python multiprocessing knjižnica. , s Interneta, `https://docs.python.org/3/library/multiprocessing.html` , kolovoz 2023.2.

[14] MacKenzie, I.S.: "Human-Computer Interaction: An Empirical Research Perspective." Morgan Kaufmann, Amsterdam (2013)

[15] MacKenzie, Ian. (2013). A Note on the Validity of the Shannon Formulation for Fitts' Index of Difficulty. Open Journal of Applied Sciences. 03. 360-368. 10.4236/ojapps.2013.36046.

[16] Within-subjects vs. Between-subjects Designs: Which to Use?, s Interneta, `https://www.yorku.ca/mack/RN-Counterbalancing.html`, kolovoz 2024.

[17] FittsStudy. , s Interneta, `https://depts.washington.edu/acelab/proj/fittsstudy/index.html` , kolovoz 2024.

[18] TextTest, s Interneta, `https://depts.washington.edu/acelab/proj/texttest/`, kolovoz 2024.

[19] V. Ecimović, "Operacija pokazivanja-i-odabira zasnovana na glasovnom upravljanju", Diplomski rad, Sveučilište u Rijeci, Tehnički fakultet, Rijeka, 2023. Dostupno na: `https://urn.nsk.hr/urn:nbn:hr:190:025397`

[20] T. M. Starbird, D. W. Runnels, and M. G. Young. *Text/Graphics Separation and Analysis Study*. NASA Technical Report, NASA Center for AeroSpace Information (CASI), 2000. Dostupno na: `https://ntrs.nasa.gov/api/citations/20000021488/downloads/20000021488.pdf`

[21] Hart, S. G. (2006). *Nasa-Task Load Index (NASA-TLX); 20 Years Later.* Proceedings of the Human Factors and Ergonomics Society Annual Meeting, 50(9), 904-908. Dostupo na: `https://doi.org/10.1177/154193120605000909`

Sažetak

U ovome radu istražuje se učinkovitost unosa teksta na stolnim računalima putem praćenja pomaka očiju. Implementirana su dva modaliteta interakcije koji se temelje na slobodnoj putanji virtualnog pokazivača te diskretnom pomaku lijevo-desno. Dizajnirane su i dvije različite virtualne tipkovnice – dok se jedna zasniva na standardnom QWERTY rasporedu, druga koristi podjelu znakova u četiri kvadranta. Organizirano je i provedeno eksperimentalno vrednovanje s ciljem uvida u vrijednosti metrika WPM i TER za sve kombinacije razina nezavisnih varijabli. Dodatno je provedeno i ispitivanje percipiranog opterećenja interakcije pomoću upitnika NASA-TLX. S obzirom na poznatu geometriju dizajniranih tipkovnica, jezični model, te eksperimentalno utvrđene vremenske značajke operacije pokazivanja-i-odabira u dotičnom kontekstu, izračunate su predviđene gornje granice brzine unosa teksta. Konačno, rezultati takvog prediktivnog modeliranja uspoređeni su s rezultatima provedenog empirijskog istraživanja.

Ključne riječi — **Interakcija čovjeka i računala, praćenje pomaka očiju, unos teksta, prediktivno modeliranje, Fittsov zakon,**

Abstract

This thesis investigates the efficiency of eye-tracking-based text entry on desktop computers. Two interaction modalities are implemented based on the free path of the virtual cursor and discrete left-right movement. Two virtual keyboards have been designed – while one is based on the standard QWERTY layout, the other uses the division of characters into four quadrants. An experimental evaluation was carried out to gain insight into the values of the WPM and TER metrics for all combinations of levels of independent variables. In addition, the perceived interaction workload was examined by utilizing the NASA-TLX questionnaire. The predicted upper limits of text entry speed were calculated given the known geometry of the designed keyboards, the language model, and the experimentally determined temporal characteristics of the point-and-select operation in the respective context. Finally, the results of such predictive modeling were compared with the results of the empirical research.

Keywords **— Human-Computer Interaction, eye tracking, text entry, predictive modeling, Fitts' Law**