НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ імені ІГОРЯ СІКОРСЬКОГО»
ФАКУЛЬТЕТ ІНФОРМАТИКИ ТА ОБЧИСЛЮВАЛЬНОЇ ТЕХНІКИ
Кафедра автоматизованих систем обробки інформації та управління

До захисту допущено:
В.о. завідувача кафедри

_____ *Олександр ПАВЛОВ*

(підпис) (вл.ім'я, прізвище)

"___"_____2021 р.

Дипломний проєкт

на здобуття ступеня бакалавра

за освітньо-професійною програмою «Інформаційні управляючі системи та технології»
спеціальності 126 «Інформаційні системи та технології»

на тему: «*Система розпізнавання жестів для вивчення дактильної абетки* »

Виконала: студентка *IV* курсу, групи *ІС-71*
Руденко Анастасія Олександрівна
(прізвище, ім'я, по батькові) (підпис)

Керівник *ст. викладач Халус Олена Андріївна*
(посада, науковий ступінь, вчене звання, прізвище, ім'я, по батькові) (підпис)

Консультант з графічної документації *доц., к.т.н. Жураковська Оксана Сергіївна*
(посада, науковий ступінь, вчене звання, прізвище, ім'я, по батькові) (підпис)

Рецензент
(посада, науковий ступінь, вчене звання, прізвище, ім'я, по батькові) (підпис)

Засвідчую, що у цьому дипломному проєкті немає запозичень з праць інших авторів без відповідних посилань.
Студентка _____
(підпис)

Київ – 2021 року

Національний технічний університет України
“Київський політехнічний інститут імені Ігоря Сікорського”

Факультет (інститут) *інформатики та обчислювальної техніки*
(повна назва)

Кафедра *автоматизованих систем обробки інформації та управління*
(повна назва)

Рівень вищої освіти – перший (бакалаврський)

Спеціальність – 126 «Інформаційні системи та технології»

Освітньо-професійна програма «Інформаційні управляючі системи та технології»

ЗАТВЕРДЖУЮ
В.о. завідувача кафедри
_____ *Олександр ПАВЛОВ*
(підпис) (вл.ім’я, прізвище)
“___”_____2021 р.

ЗАВДАННЯ
на дипломний проєкт студенту

Руденко Анастасії Олександрівні
(прізвище, ім’я, по батькові)

1. Тема проєкту « *Система розпізнавання жестів для вивчення дактильної абетки* »,
керівник проєкту *Халус Олена Андріївна, ст. викладач*
(прізвище, ім’я, по батькові, науковий ступінь, вчене звання)
затверджені наказом по університету від “*11*” *05* 2021 р. № 1139-с

2. Термін подання студентом проєкту “*04*”*червня* 2021 року

3. Вихідні дані до проєкту
Технічне завдання

4. Зміст пояснювальної записки
1. Загальні положення: основні визначення та терміни, опис предметного середовища, огляд ринку програмних продуктів, постановка задачі
2. Інформаційне забезпечення: вхідні дані, вихідні дані, опис структури бази даних
3. Математичне забезпечення: змістовна та математична постановки задачі, обґрунтування та опис методу розв’язання

4. Програмне та технічне забезпечення: засоби розробки, вимоги до технічного забезпечення, архітектура програмного забезпечення, побудова звітів

5. Технологічний розділ: керівництво користувача, методика випробувань програмного продукту

5. Перелік графічного матеріалу

1. Схема структурна варіантів використань

2. Схема структурна послідовності

3. Схема структурна компонентів

4. Креслення вигляду екранних форм

5. Рішення з математичного забезпечення

6. Схема структурна діяльності

6. Консультанти розділів проєкту

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв

7. Дата видачі завдання «*7*» *квітня 2021 року*

Календарний план

№ з/п	Назва етапів виконання дипломного проєкту	Термін виконання етапів проєкту	Примітка
1.	*Вивчення рекомендованої літератури*	*09.04.2021*	
2.	*Аналіз існуючих методів розв'язання задачі*	*12.04.2021*	
3.	*Постановка та формалізація задачі*	*20.04.2021*	
4.	*Розробка інформаційного забезпечення*	*23.04.2021*	
5.	*Алгоритмізація задачі*	*28.04.2021*	
6.	*Обґрунтування використовуваних технічних засобів*	*30.04.2021*	
7.	*Розробка програмного забезпечення*	*08.05.2021*	
8.	*Налагодження програми*	*10.05.2021*	
9.	*Виконання графічних документів*	*13.05.2021*	
10.	*Оформлення пояснювальної записки*	*14.05.2021*	
11.	*Подання ДП на попередній захист*	*14.05.2021*	
12.	*Подання ДП на основний захист*	*04.06.2021*	
13.	*Подання ДП рецензенту*	*07.06.2021*	

Студент Анастасія РУДЕНКО

Керівник Олена ХАЛУС

№з/п	Формат	Позначення	Найменування	Кількість листів	Примітка
1	А4		Завдання на дипломний проєкт	2	
2	А4	ДП 7123.00.000 ПЗ	Пояснювальна записка	73	
3	А4	ДП 7123.01.000 ТЗ	Технічне завдання	11	
4	А3	ДП 7123.02.000 ССВ	Схема структурна варіантів використань	1	
5	А3	ДП 7123.03.000 ССП	Схема структурна послідовності	1	
6	А3	ДП 7123.04.000 СК	Схема структурна компонентів	1	
7	А3	ДП 7123.05.000 КЕ	Креслення вигляду екранних форм	1	
8	А3		Рішення з математичного забезпечення	1	
8	А3	ДП 7123.06.000 ССД	Схема структурна діяльності	1	

Зм.	Арк.	ПІБ	Підп.	Дата	ДП 7123 00.000.00		
Розробн.		Руденко А.О.			Відомість дипломного проєкту	Літ.	Лист 1 / Листів 1
Керівн.		Халус О.А.					
Консульт.						КПІ ім. Ігоря Сікорського Каф. АСОІУ Гр. ІС-71	
Н/контр.		Жураковська О.С					
В.о.зав.каф.		Павлов О.А.					

Пояснювальна записка
до дипломного проєкту

на тему: « *Система розпізнавання жестів для вивчення дактильної абетки* »

Київ – 2021 року

АНОТАЦІЯ

Структура та обсяг роботи. Пояснювальна записка дипломного проекту складається з п’яти розділів, містить 26 рисунків, 8 таблиць, 1 додаток, 5 джерел.

Дипломний проект присвячений розробці системи розпізнавання жестів для вивчення дактильної абетки.

Розділ загальних положень включає в себе опис предметного середовища, огляд наявних аналогів та постановку задачі. Також цей розділ включає опис діяльності та функціональних складових системи.

У розділі інформаційного забезпечення були виявлені вхідні та вихідні дані.

Розділ математичного забезпечення присвячений огляду наявних методів розробки програмної реалізації.

У розділі програмного забезпечення вказано діаграми послідовності, класів, компонентів, специфікацію функцій. Також представлено опис архітектури системи, програмного та апаратного забезпечення.

У технологічному розділі наведено керівництво користувача, опис випробувань програмної реалізації.

КЛЮЧОВІ СЛОВА: ВЕБ-ЗАСТОСУНОК, СИСТЕМА, РОЗПІЗНАВАННЯ ЖЕСТІВ, REACT.JS, TENSORFLOW.JS

ABSTRACT

Structure and scope of work. The explanatory note of the diploma project consists of five sections, contains 26 drawings, 8 tables, 1 application, 5 sources.

Diploma project dedicated to the development of system recognition of gestures to explore dactyl alphabet.

The section of terms includes a description of the subject environment, an overview of existing analogues and formulation of the problem. Also, this section includes a description of the activities and functional components of the system.

Input and output data were found in the information support section.

The section of mathematical software is devoted to the review of available methods of software implementation development.

In the software section specified sequence diagrams, classes, components, data functions. There is also a description of the system architecture, software and hardware.

Technological section provides a user manual, description of software implementation tests.

KEY WORDS: WEB APPLICATION, SYSTEM, GESTURE RECOGNITION, REACT.JS, TENSORFLOW.JS

ЗМІСТ

ВСТУП

Українська дактильна абетка являє собою допоміжну систему української жестової мови. За допомогою жестів пальцями, найчастіше, однієї руки, що зігнута у лікті, можна показати літеру українського алфавіту. Абетка, зазвичай, використовується для прояснення слів, які були незрозумілими для співрозмовника, або таких, які не мають відповідного жестового позначення. Оскільки, вивчення дактильної абетки є набагато легшим за вивчення мови жестів, людина, без знання останньої, зможе, за потреби, комунікувати із співрозмовником з вадами слухового та/або мовного апарату.

Наразі, невідомо хто саме першим винайшов дактильну абетку, але перші її згадки датовані X ст. у латинській Біблії. На думку дослідників, дактильна абетка була вперше опублікована іспанським ченцем у 1593 р. Українська жестова мова утворилася від французької та австро-угорської мови у 1805 році та належить до сім'ї французької жестової мови.

У сучасному світі вже зараз більш ніж 5% населення світу мають проблему втрати слуху. Згідно з прогнозами, до 2050 року майже 2,5 мільярда чоловік будуть страждати від проблем зі слухом у тій чи іншій мірі і щонайменше 700 мільйонів чоловік потребуватимуть реабілітаційних послугах у зв'язку з втратою слуху. Більше 70 мільйонів людей по всьому світі використовують ту чи іншу мову жестів.

Метою розробки є створення системи розпізнавання жестів для вивчення дактильної абетки, для поліпшення умов соціалізації людей з вадами слухового чи/або мовного апарату.

1 ЗАГАЛЬНІ ПОЛОЖЕННЯ

1.1 Опис предметного середовища

Система розпізнавання жестів для вивчення дактильної абетки це інтерактивне веб-застосування, головною ціллю створення якого є поліпшення умов соціалізації людей з вадами слухового чи/або мовного апарату.

Людина, яка намагається вивчити дактильну абетку самостійно, потребує контролю за правильністю відтворення жесту у реальному часі. Задля цього система розпізнавання жестів для вивчення дактильної абетки використовує алгоритми розпізнавання жестів з відео-потоку у реальному часу. Тож, процес навчання відбувається по моделі завдання – виконання – оцінка (жест відтворено/не відтворено).

Цей веб-застосунок може бути корисним для будь-якої людини, яка має мету вивчити дактильну абетку та має базові навички користування комп’ютером.

1.1.1 Опис процесу діяльності

В даній роботі об’єктом автоматизації є процес вивчення дактильної абетки.

Вхідні дані представлені у вигляді відео-потоку з камери пристрою користувача. Користувач, знаходячись перед веб-камерою, відтворює жест-завдання.

Результатом роботи системи є оцінка відтворення жесту користувачем (відтворено/не відтворено), отримана після співставлення жесту користувача з жестом-завданням.

1.1.2 Опис функціональної моделі

Користувач системи – це людина, яка має бажання вивчити дактильну абетку. При використанні системи, користувачу надається можливість:

а) вибору букви (жесту) для відтворення;

б) отримання завдання (жесту) від системи;

в) отримання оцінки правильності виконання жесту.

На рисунку 1.1 наведено діаграму використання системи розпізнавання жестів для вивчення дактильної абетки.

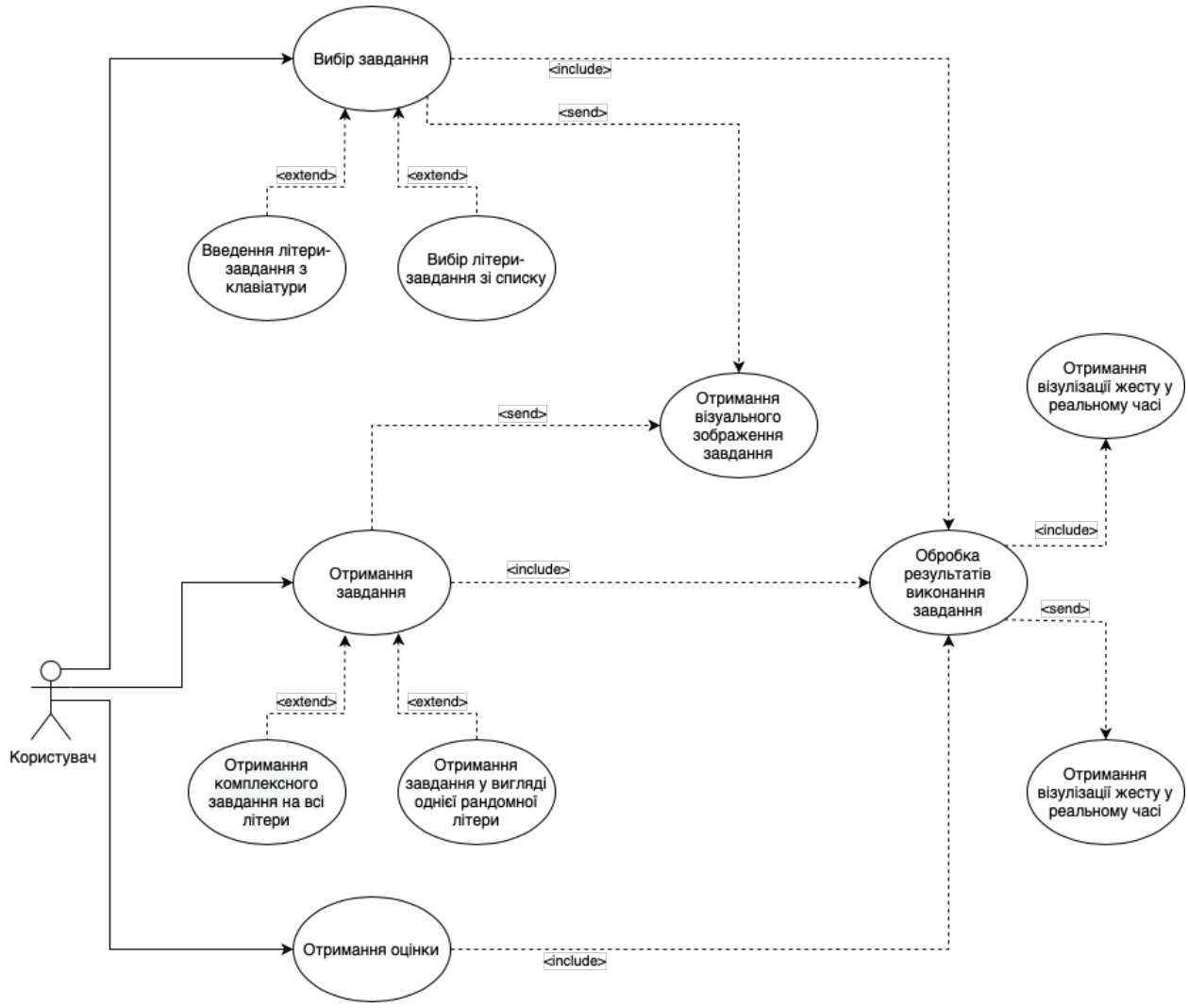


Рисунок 1.1 – Схема структурна варіантів використання

Система розпізнавання жестів для вивчення дактильної абетки повинна відповідати наступним функціональним вимогам:

а) відтворення відео-потоку з камери комп’ютеру;
б) надання системою завдання (жесту) для відтворення користувачем;
в) вибір користувачем завдання (жесту) для відтворення користувачем;
г) відображення відповідного учбового матеріалу (зображення жесту) до завдання;
ґ) розпізнавання однієї людської долоні з відео-потоку;
д) відображення 21 ключової точки руки у реальному часі;
е) розпізнавання жесту, який показано за допомогою однієї долоні;
є) надання оцінки про виконання користувачем потрібного жесту.

1.2 Огляд наявних аналогів

У процесі аналізу існуючих систем для вивчення дактильної мови жестів було виокремлено такі типи аналогів:

а) системи, які у процесі навчання оцінюють виконання жесту;
б) системи, які у процесі навчання не оцінюють виконання жесту.

Системи першого типу найбільш поширені. Вони представлені у вигляді мобільних застосувань чи веб-застосувань. До їх переваг можна віднести гейміфікований процес навчання, що заохочує користувача. Велику кількість додаткових матеріалів для вивчення (відео, фото, текстові ресурси). Також такі програми потребують менше пам’яті та ресурсів пристроїв, на яких вони використовуються.

Недоліками систем першого типу є платна підписка та, насамперед, відсутність перевірки правильності виконання завдання користувачем.

Системи другого типу є менш поширеними, але, здебільшого, більш функціонально досконалими. Вони надають можливість користувачу не тільки побачити як потрібно правильно показувати конкретний жест, але і слідкують за правильністю його виконання. Найчастіше ця процедура перевірки виконується шляхом виявлення жестів з відео-потоку, що транслює дії користувача за допомогою камери пристрою (ноутбуку, телефону, ін.).

Системи такого типу зазвичай мають усі переваги систем першого типу. До їх недоліків відносяться: платна підписка (як правило, дорожча ніж у системах першого типу), потреба у значно більших об’ємах пам’яті та ресурсів пристроїв, на яких вони використовуються.

Прикладом першого типу аналогів є Ace ASL, який можна побачити на рисунку 1.2.

Це мобільне застосування, яке було створено компанією Sign All. Проект став доступним для користувачів IOS пристроїв у квітні 2021 року. За допомогою розробленої технології, яка використовує штучний інтелект та комп’ютерний зір, мобільний додаток здатний розпізнавати жести, показані користувачем. Це можливо із використанням аналізу форми, напряму, рухів та розташування руки. Користувач має оформити підписку, щоб мати змогу користуватися програмою.

Також, потрібно виділити деякі переваги, серед яких:

а) гейміфікація процесу навчання;

б) можливість вивчити американську абетку та числа мовою жестів;

в) використовує базу даних американської жестової мови (ASL);

г) у додатку можна подивитися навчальні відео.

Серед недоліків:

а) більша частина програми доступна лише за платною підпискою: на місяць $3,99/ повний доступ на необмежений час $9,99 - для одного користувача;

б) для вивчення доступна лише англійська мова жестів;

в) додаток працює лише на IOS пристроях.

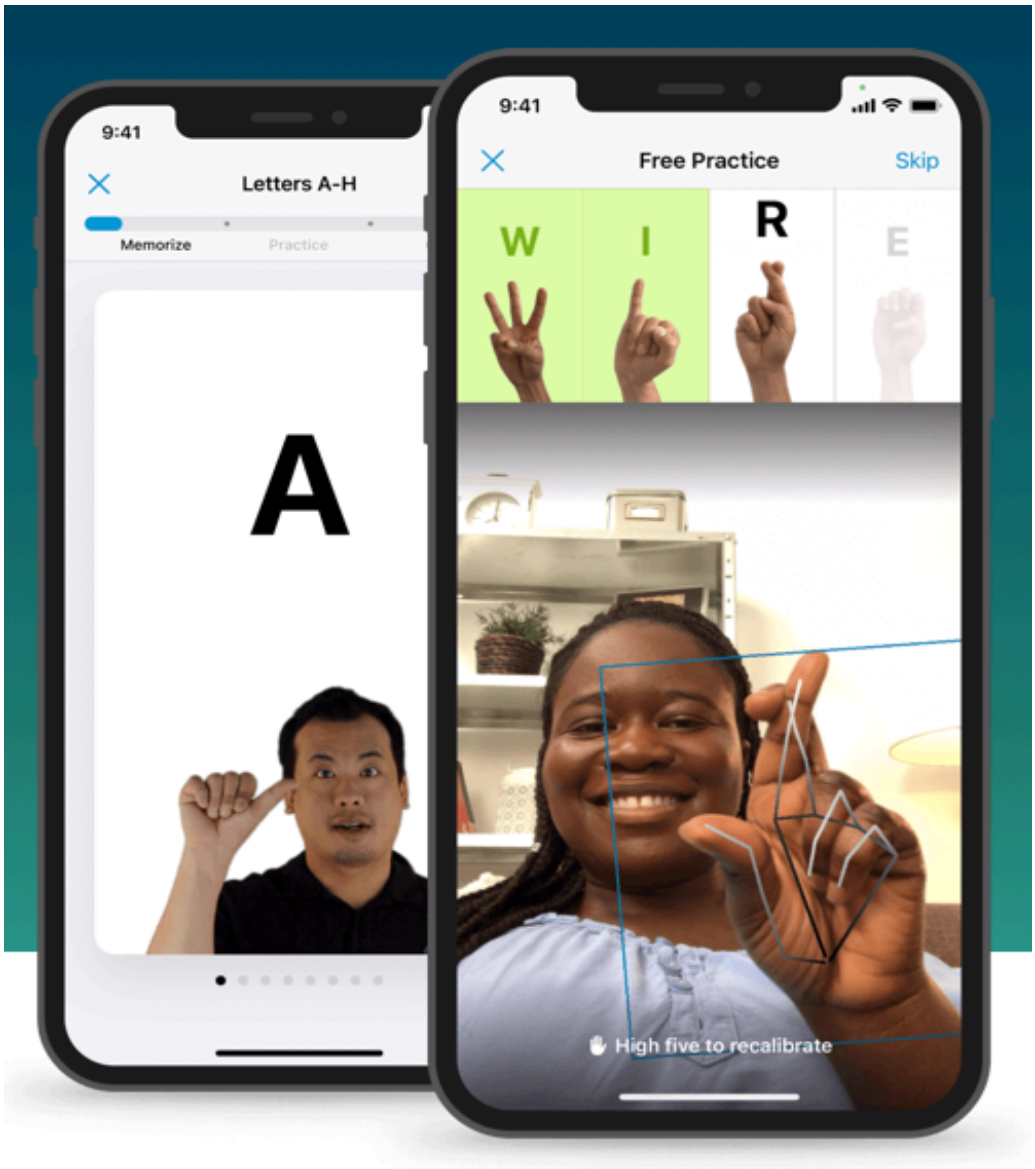


Рисунок 1.2 – Приклад роботи додатку Ace ASL

Першим прикладом другого типу аналогів може слугувати мобільний додаток Sign Language ASL, який можна побачити на рисунку 1.3. Рейтинг програми у Play Market – 4.7/5. Кількість завантажень більше 100 000.

Додаток дає змогу вивчити алфавіт американської мови жестів (ASL), допомагає навчитися використовувати та перекладати мову жестів за допомогою інтерактивних відео уроків.

Серед інших переваг потрібно виділити:

а) наявність поширених фраз та привітань, що використовуються щодня;

б) дозволяє вивчити основні фрази мови жестів.

Серед недоліків:

а) відсутня можливість перекладу мови жестів;

б) відсутня можливість перевірки правильності використання жестової мови користувачем;

в) тільки перші 2 тижні є безкоштовними, у подальшому користувач може придбати різні тарифні плани: на місяць $5.99/ рік $59.99;

г) відсутня українська дактильна абетка.

Рисунок 1.3 – Приклад роботи програми Sign Language ASL

Другим прикладом другого типу аналогів є сервіс Spread the Sign від некомерційної асоціації «Європейський центр жестових мов», який можна побачити на рисунку 1.4.

Цей веб-сайт доступний на 42 мовах, одна з яких українська. У ньому зібрано варіації жестів з різних жестових мов по всьому світу. Він надає змогу, на прикладі GIF відео, побачити переклад великої кількості слів або речень. Наприклад, як зображено на рисунку 1.5, слово “Привіт” можна

побачити у 33 різних мовах жестів різних країн, включаючи українську.

Тож, серед переваг потрібно виділити:

а) переклад на мову жестів здійснюється за допомогою відео;

б) присутній словник жестової мови різних країн, можна перекладати як окремі слова, так і популярні речення;

в) на сайті є можливість змінювати швидкість програвання відео;

г) присутня українська жестова мова;

ґ) сервіс є безкоштовним.

Основним недоліком є відсутність можливості перевірки правильності використання жестової мови користувачем.

Рисунок 1.4 – Spread the Sign логотип

Рисунок 1.5 – Spread the Sign приклад роботи програми

З огляду на виявлені переваги та недоліки існуючих аналогів, була розроблена система розпізнавання жестів для вивчення дактильної абетки з можливістю оцінки правильності виконання жесту.

1.3 Постановка задачі

Система розпізнавання жестів для вивчення дактильної абетки це інтерактивний веб-застосунок, ціллю створення якого є поліпшення умов соціалізації людей з вадами слухового чи/або мовного апарату.

1.3.1 Призначення розробки

Наразі існує безліч додатків, які дозволяють користувачам вивчати іноземні мови. Вони дозволяють практикувати вимову та отримувати негайні відгуки від програми. Але така функціональність майже не представлена для вивчення дактильної абетки або мови жестів.

Система розпізнавання жестів для вивчення дактильної абетки відкриває можливість інтерактивного процесу навчання, що є важливим етапом на шляху до підвищення якості комунікації між усіма людьми.

Дана розробка призначена для поліпшення умов соціалізації людей з вадами слухового чи/або мовного апарату.

1.3.2 Цілі та задачі розробки

Головною метою розробки є створення системи розпізнавання жестів для вивчення дактильної абетки.

Цілями розробки є:

а) покращення умов навчання для людей з вадами слухового чи/або мовного апарату;

б) зменшення часу, який потрібен для вивчення української дактильної абетки.

Для досягнення мети розробки системи розпізнавання жестів для вивчення дактильної абетки наступні задачі потрібні бути реалізованими:

а) можливість відтворення відео-потоку з камери комп’ютера;

б) можливість формування завдання для користувача:

1) вибір завдання (жесту) системою:

- отримання комплексного завдання на всі літери;
- отримання завдання у вигляді однієї рандомної літери;

2) вибір завдання (жесту) користувачем:

- введення літери-завдання з клавіатури;
- вибір літери-завдання зі списку;

в) відображення учбового матеріалу у відповідності до завдання;

г) розпізнавання однієї людської долоні з відео-потоку;

г) відображення 21 ключової точки руки у реальному часі на відео;

д) розпізнавання жесту, відтвореного користувачем за допомогою однієї долоні;

е) надання оцінки про виконання користувачем потрібного жесту.

Висновок до розділу

В даному розділі було досліджено предметне середовище та підстави розробки системи розпізнавання жестів для вивчення дактильної абетки.

Був описаний алгоритм, що буде автоматизовано, та визначені можливості користувача при роботі із застосунком. Проведено аналіз найпопулярніших аналогів, які допомагають користувачу у вивченні дактильної абетки. На підставі даного аналізу було прийняте рішення щодо розробки системи, яка оцінює виконання завдання (жесту).

Визначені функціональні вимоги, яким має відповідати система розпізнавання жестів для вивчення дактильної абетки.

2 ІНФОРМАЦІЙНЕ ЗАБЕЗПЕЧЕННЯ

2.1 Вхідні дані

Вхідними даними є відео-потік з камери пристрою користувача, який оброблюється інструментами бібліотеки Tensorflow.js MediaPipe Handpose. Зображення руки буде отримано за умови, що користувач, який перебуває перед веб-камерою, відтворює жест-завдання. Відео отримується за допомогою інструментів бібліотеки react-webcam. На рисунку 2.1 зображено фрагмент відео-потоку з вже розміченими ключовими точками долоні користувача.

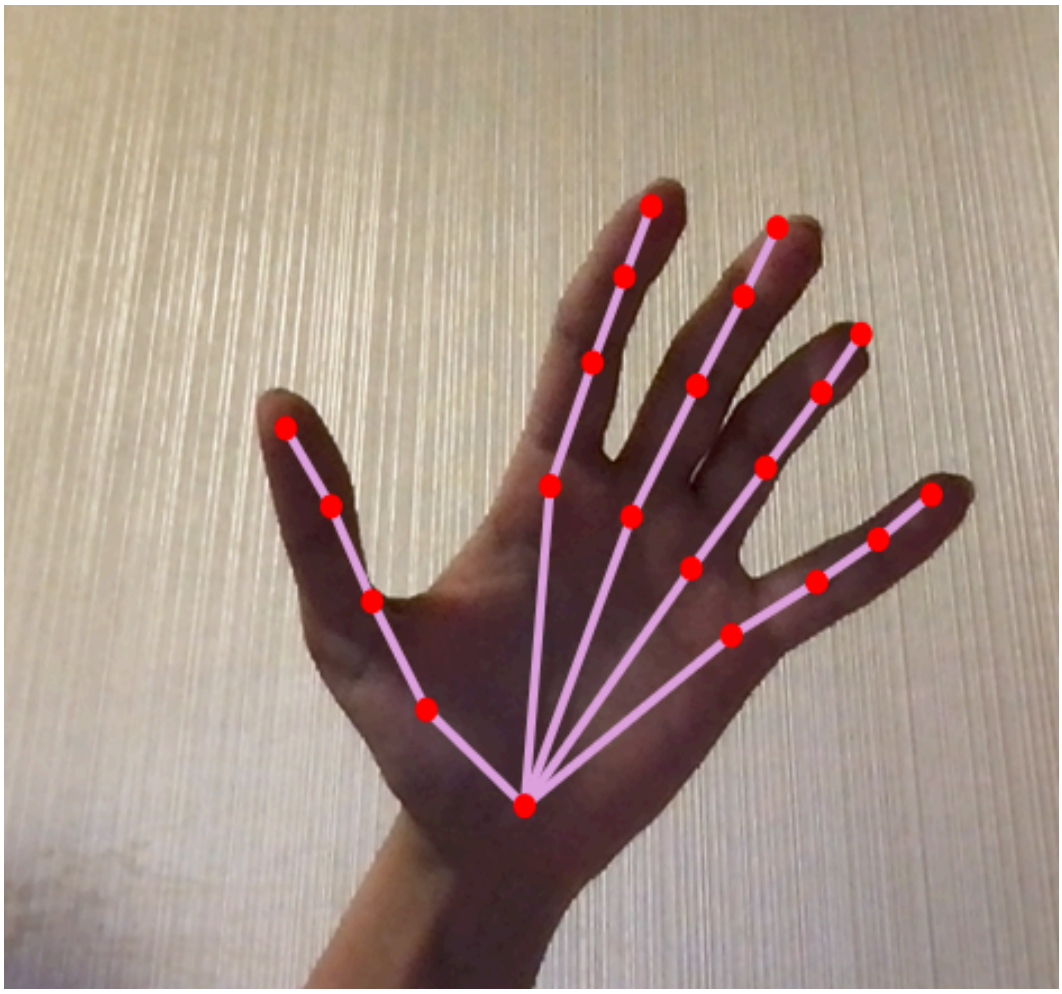


Рисунок 2.1 – Приклад відео-потоку у якості вхідних даних програми

Іншим видом вхідних даних можна вважати завдання (жест), що обирає користувач системи.

2.2 Вихідні дані

Вихідними даними є оцінка системи, після опрацювання вхідних даних (відео-потоку з жестом). Оцінка надається у формі повідомлення, що з’являється на сторінці веб-застосування, як зображено на рисунку 2.2.

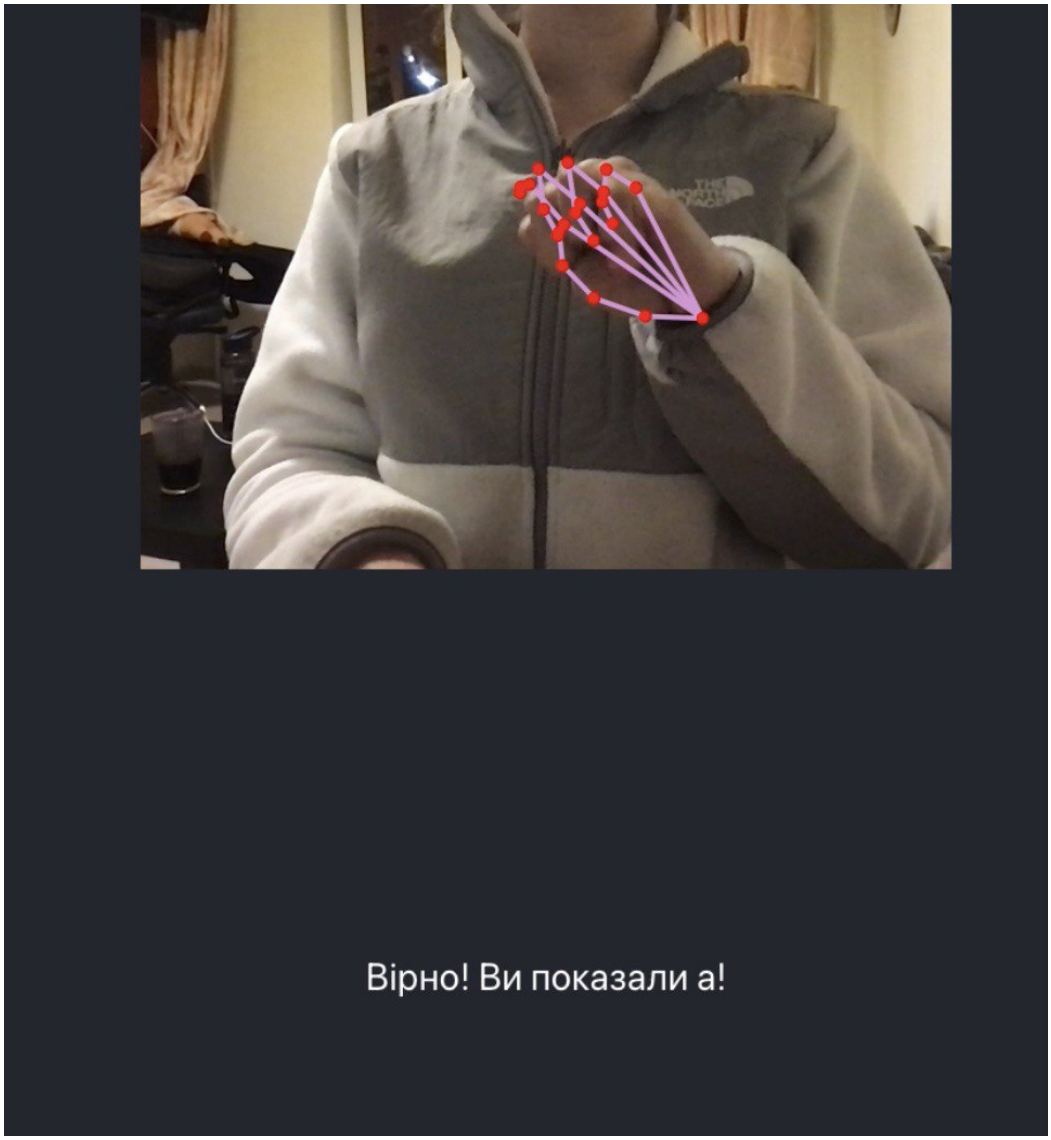


Рисунок 2.2 – Приклад надання оцінки виконання жесту системою

2.3 Структура масивів інформації

Після естімації відео-потоку бібліотекою Tensorflow.js MediaPipe Handpose, формується масив інформації, який описує положення руки у конкретний момент часу. Даний масив зберігається у вигляді об’єкту hand, який зображено на рисунку 2.3.

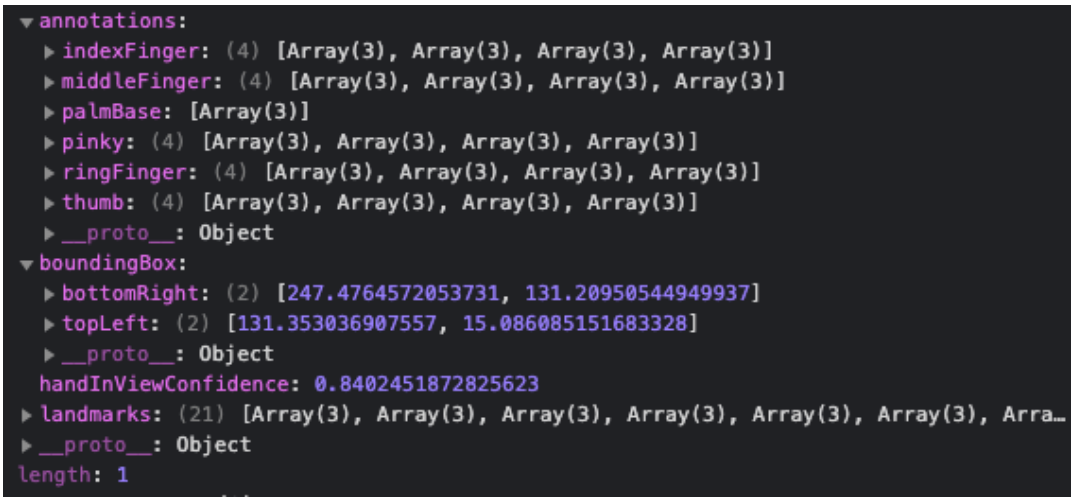


Рисунок 2.3 – Інформація про положення руки (hand)

Нижче у таблиці 2.1 наведено опис інформації, з якої складається об’єкт hand.

Таблиця 2.1 – Об’єкт даних hand

Складова	Опис складової об’єкту
annotations	Інформація про положення пальців і основи долоні. Включає в себе indexFinger, middleFinger, palmBase, pinky, ringFinger, thumb (рисунок 2.4).
boundingBox	Інформація про положення рамки (фрейму), у якій зафіксовано долоню.
landmarks	Інформація про положення кожної з 21 значущої точки, що описують долоню.

```
▼ hand: Array(1)
  ▼ 0:
    ▼ annotations:
      ▼ indexFinger: Array(4)
        ▶ 0: (3) [205.9020594584456, 98.56653860631354, -26.656089782714844]
        ▶ 1: (3) [224.60366229822748, 84.71570933776204, -17.709741592407227]
        ▶ 2: (3) [219.42711166977156, 79.60491525822582, -4.839853763580322]
        ▶ 3: (3) [213.41249612501383, 79.43852028899872, -0.5860787630081177]
          length: 4
        ▶ __proto__: Array(0)
      ▼ middleFinger: Array(4)
        ▶ 0: (3) [205.0122823555682, 89.14407382144628, -27.58536720275879]
        ▶ 1: (3) [219.32364696884287, 74.30479509812047, -19.419822692871094]
        ▶ 2: (3) [215.01553552230297, 71.63687503013563, -6.466494083404541]
        ▶ 3: (3) [209.73402749669629, 71.76846946718126, -2.8753271102905273]
          length: 4
        ▶ __proto__: Array(0)
      ▼ palmBase: Array(1)
        ▶ 0: (3) [154.2258318147026, 76.64713334164055, -0.001140601933002472]
          length: 1
        ▶ __proto__: Array(0)
      ▼ pinky: Array(4)
        ▶ 0: (3) [194.54510602065898, 65.01234679259272, -23.765430450439453]
        ▶ 1: (3) [206.03198038906604, 56.217874278431864, -16.196134567260742]
        ▶ 2: (3) [205.2583642322167, 56.201966430109906, -12.735770225524902]
        ▶ 3: (3) [202.93733171585134, 56.30094436661287, -13.072440147399902]
          length: 4
        ▶ __proto__: Array(0)
      ▼ ringFinger: Array(4)
        ▶ 0: (3) [200.29634287042674, 77.00663858296073, -25.85840606689453]
        ▶ 1: (3) [213.53769417599517, 64.54746169154905, -16.585355758666992]
        ▶ 2: (3) [210.86904503220293, 63.6906029645315, -8.566932678222656]
        ▶ 3: (3) [206.68966784024053, 63.84860015752892, -7.841882228851318]
          length: 4
        ▶ __proto__: Array(0)
      ▼ thumb: Array(4)
        ▶ 0: (3) [170.4570815752796, 91.4052280890235, 2.239788055419922]
        ▶ 1: (3) [189.36737850048888, 96.28502527518107, -1.6979652643203735]
        ▶ 2: (3) [205.8680956306923, 93.20868248801168, -4.143738746643066]
        ▶ 3: (3) [217.93378664714933, 89.87640183615676, -5.7298994064331055]
          length: 4
        ▶ __proto__: Array(0)
      ▶ __proto__: Object
```


Рисунок 2.4 – Дані про положення руки у конкретний момент часу, сформовані по пальцях

Дані, для розпізнавання жестів дактильної абетки зберігаються у JavaScript файлах і називаються у форматі <українська літера англійською мовою>_<номер літери у абетці> .js. як показано на рисунку 2.5.

Рисунок 2.5 – Приклад назв файлів, що використовуються для класифікації жестів

В цих файлах кожен жест розбивається на пальці. Для кожного пальцю описуються правила, які його характеризують, такі як напрямок та згин. Також вказується прийнятна ймовірність. Приклад опису жесту представлено на рисунку 2.6.

```
// import dependencies
import {Finger, FingerCurl, FingerDirection, GestureDescription} from 'fingerpose';

// define gesture description
export const bSign2 =  new GestureDescription('6');

// thumb
bSign2.addCurl(Finger.Thumb, FingerCurl.HalfCurl, 1.0)
bSign2.addDirection(Finger.Thumb, FingerDirection.DiagonalUpLeft , 1.0)
bSign2.addDirection(Finger.Thumb, FingerDirection.DiagonalUpRight, 1.0)

// index
bSign2.addCurl(Finger.Index, FingerCurl.NoCurl, 1.0)
bSign2.addDirection(Finger.Index, FingerDirection.VerticalUp, 1.0)

// middle
bSign2.addCurl(Finger.Middle, FingerCurl.FullCurl, 1.0)
bSign2.addCurl(Finger.Middle, FingerCurl.HalfCurl, 0.75)
bSign2.addDirection(Finger.Middle, FingerDirection.VerticalUp, 1.0)
bSign2.addDirection(Finger.Middle, FingerDirection.DiagonalUpLeft, 0.75)
bSign2.addDirection(Finger.Middle, FingerDirection.DiagonalUpRight, 0.75)

// ring
bSign2.addCurl(Finger.Ring, FingerCurl.FullCurl, 1.0)
bSign2.addDirection(Finger.Ring, FingerDirection.VerticalUp, 1.0)

// pinky
bSign2.addCurl(Finger.Pinky, FingerCurl.FullCurl, 1.0)
bSign2.addDirection(Finger.Pinky, FingerDirection.DiagonalUpLeft, 1.0)
bSign2.addDirection(Finger.Pinky, FingerDirection.DiagonalUpRight, 1.0)
```

Рисунок 2.6 – Приклад файлу, для класифікації жестів

Висновок до розділу

Даний розділ описує вхідні та вихідні дані для системи розпізнавання жестів для вивчення дактильної абетки. У розділі наведено приклад вхідних даних (відео-потік). Також продемонстровано вихідні данні , а саме оцінка виконання завдання (жесту) користувачем.

Окрім цього визначено структуру масивів інформації, яка використовується для розпізнавання жестів.

3 МАТЕМАТИЧНЕ ЗАБЕЗПЕЧЕННЯ

3.1 Змістовна постановка задачі

Користувач системи на камеру свого комп'ютеру показує жест, відтворення якого запрошує система або жест, який був обраний для відтворення самим користувачем. Для підтвердження того, що жест відтворений користувачем дійсно є жестом-завданням, необхідно виконати наступні підзадачі:

а) розпізнання долоні у відео-потоці;

б) визначити положення долоні у відео-потоці;

в) виокремити 21 ключову точку долоні та визначити їх координати;

г) обробити ключові точки і в результаті отримати інформаційний об'єкт hand;

ґ) надати оцінку, чи відповідає інформаційний об'єкт hand жесту завданню.

3.2 Математична постановка задачі

Нехай маємо, отриманий з відео, фрейм, у якому міститься долоня. Вона характеризується 21 ключовою точкою, кожна з яких визначається трьома координатами $(x_i, y_i, z_i), i = \overline{0,20}$. Також маємо вже описані набори $(x_i, y_i, z_i), i = \overline{0,20}$, які відповідають жестам української дактильної абетки.

Головна задача – при постійній отриманні нових наборів координат ключових точок порівняти їх значення зі значеннями ключових точок жесту-завданню і надати або позитивну оцінку коли вони співпадають не менше ніж на 80%, або негативну у разі не співпадіння.

3.3 Обґрунтування методу розв’язання

Для вирішення подібних задач зазвичай використовуються програмні рішення з області машинного зору, такі як OpenCV та DLib.

OpenCV це проект з відкритим кодом - бібліотека алгоритмів комп'ютерного зору. DLib, в свою чергу, – це бібліотека інсструментів для машинного навчання.

Це дуже потужні інструменти і вони є одними з найпопулярніших у світі для вирішення подібних задача, але більшість способів використання пов’язані більш з дослідницькою діяльністю, і потребує використання лише мови програмування Python. Можливі проблеми обумовлені відносно великими труднощами, у разі потреби використовувати додаткові мови програмування, наприклад, для написання інтерфейсу користувача. А написання веб-застосунку на мові програмування Python є складним процесом і сьогодні ця мова не є дуже популярним рішенням для написання веб-застосунків або, наприклад, мобільних додатків.

Іншим варіантом вирішення подібних задач є використання спеціальних бібліотек, як, наприклад, TensorFlow.js - бібліотеку з відкритим кодом, призначенням якої є визначення, навчання і запуску моделей машинного навчання повністю в браузері, використовуючи Javascript і високорівневе API. Ця бібліотека є новим проєктом, який вже зараз отримав багато зацікавлених користувачів, саме через свою малу у порівнянні з іншими вагу та гнучкість.

Значною перевагою TensorFlow.js є те, що немає необхідності встановлювати будь-які бібліотеки або драйвера. Потрібно просто відкрити веб-сторінку, і програма готова до запуску.

Також важливо відмітити велику кількість вже тренованих моделей, які доступні для використання у проектах.

3.4 Опис методів розв’язання

Для розв’язання задач розробки було обрано бібліотеку для навчання і запуску моделей машинного навчання повністю в браузері TensorFlow.js.

Першим кроком є розпізнавання долоні, положення якої визначається моделлю , після чого повертається обмежувальне поле, що є орієнтованим (фрейм). Розташування фрейму зберігається в об’єкт hand у вигляді наборів з двох координат для нижнього правого та верхнього лівого кута, дивись Рисунок 3.1.

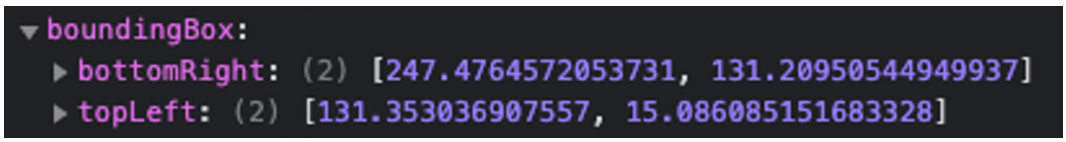


Рисунок 3.1 – Фрейм з рукою

Після цього проводиться виокремлення ключових точок тільки на вже підготовленому фреймі (області зображення) та повертаються тривимірні координати цих точок. Визначається 21 ключова точка (дивись Рисунок 3.2), які після оброблюються і перевіряються на відповідність наявним жестам, які попередньо описані за допомогою інструментів бібліотеки fingerpose.

Рисунок 3.2 – Візуалізація відрізків, для визначення положення руки

Для опису жесту ключові точки формують пальці. Кожен палець складається з 4 ключових точок і окремо фіксується основа долоні. Після чого формується скелет долоні.

```
// [0]      Palm
// [1-4]    Thumb
// [5-8]    Index
// [9-12]   Middle
// [13-16] Ring
// [17-20] Pinky
pointsMapping: {
  0: [[0, 1], [1, 2], [2, 3], [3, 4]],
  1: [[0, 5], [5, 6], [6, 7], [7, 8]],
  2: [[0, 9], [9, 10], [10, 11], [11, 12]],
  3: [[0, 13], [13, 14], [14, 15], [15, 16]],
  4: [[0, 17], [17, 18], [18, 19], [19, 20]]
},
```

Рисунок 3.3 – Фрагментація ключових точок на пальці

Для точного визначення жесту дактильної абетки потрібно описати стан кожного з пальців такими можливими параметрами:

а) наявність згину:
   1) прямий;
   2) напівзігнутий;
   3) повністю зігнутий;

б) напрямок:
   1) по вертикалі вгору;
   2) по вертикалі вниз;
   3) по горизонталі вліво;
   4) по горизонталі вправо;
   5) по діагоналі вгору вправо;
   6) по діагоналі вгору вліво;
   7) по діагоналі вниз вправо;
   8) по діагоналі вниз вліво.

При використані будь-якого з параметрів для опису стану пальця, необхідно також вказати ймовірність відповідності положення пальцю (яке взято з фрейму з відео-потоку) до заданих параметрів описаних у системі жестів (0.0-1.0).

Можливо задати декілька різних значень параметрів з різними ймовірностями для одного пальця задля більш точного розпізнавання жесту. Наприклад, коли людина показує літеру «Г», дивись на Рисунок 3.2, її великий палець може з фізіологічних причин бути напрямленим не тільки горизонтально вправо або вліво (відповідно до руки, якою користувач показує жест), а й по діагоналі вниз вправо або по діагоналі вниз вліво. Цю проблему можна вирішити задавши чотири параметри напрямку: горизонтально та по діагоналі донизу, як для правої, так і до лівої руки.

Рисунок 3.4 – Літера «Г» української дактильної абетки

Розпізнавання жесту відбувається шляхом порівняння оцінених параметрів (наявність згину та напрямку) для поточного значення долоні з існуючими у система описами жестів. Якщо з ймовірністю збігу більше 80% поточне значення співпадає з описом жесту, можна казати, що система розпізнала жест. Нижче наведено алгоритм розпізнавання жесту покроково. Схематично алгоритм зображено на рисунку 3.3.

КРОК 1. Розпізнати долоню використовуючи функцію estimateHands(video) яка приймає як параметр об’єкт відео-потоку.

КРОК 1.1. Якщо на відео виявлена долоня, сформувати орієнтовний фрейм bounding box навколо долоні.

КРОК 1.2. У фреймі розпізнати розташування ключових точок долоні.

КРОК 1.2. Сформувати масив об’єктів hand.

КРОК 1.2.1. Для hand[0] об’єднати ключові точки долоні у пальці.

КРОК 2. Порівняти положення ключових точок hand[0] з положенням ключових точок, що відповідають літері-завданню використовуючи функцію.

КРОК 2.1. Ініціалізувати об’єкт GE, передавши параметром масив з усіх описаних у системі жестів.

КРОК 2.2. Розпізнати жести та зберегти у масив gesture.gestures викликавши метод GE.estimate(hand[0].landmarks, 8). Параметри відповідно: ключові точки долоні користувача у поточний момент та допустима точність рорпізнавання (80%).

КРОК 2.3. Якщо gestures.length() > 0, шукаємо індекс жесту з максимальною точністю розпізнавання, використовуючи функцію Math.max.apply(), яка знаходить максимум, і за допомогою функції indexOf() знаходимо відповідний максимуму індекс maxConfidence елементу у масиві gesture.gestures. Фіксуємо, що користувач показав жест gesture.gestures[maxConfidence].

КРОК 2.4. Якщо gesture.gestures[maxConfidence].name == GE.gestures[numberOfTaskLetter].name жест-завдання відтворено.

Рисунок 3.5 – Алгоритм розпізнавання жесту для формування оцінки виконання жесту-завдання

Висновок до розділу

В даному розділі була сформульована змістовна постановка задачі та описана математичної постановки задачі. Також у розділі охарактеризовані основні існуючи методи розв’язання подібних задач, після опису їх вад і переваг було вирішено використовувати TensorFlow.js для реалізації методу розв’язання.

4 ПРОГРАМНЕ ТА ТЕХНІЧНЕ ЗАБЕЗПЕЧЕННЯ

4.1 Засоби розробки

На підготовчому етапі було проаналізовано існуючі технології, що використовуються для вирішення поставлених задач. Було обрано мову програмування JavaScript, як основну мову для написання системи розпізнавання жестів для вивчення дактильної абетки. Оскільки вище зазначена система спроектована як веб-додаток, для реалізації проекту була використана найсучасніша бібліотека для створення користувацьких інтерфейсів – React.js, мова розмітки гіпертексту HTML та для візуальної презентації – CSS.

Процес розпізнавання людської долоні реалізовано за допомогою бібліотеки TensorFlow.js для машинного навчання частиною якої є MediaPipe Handpose бібліотекаа, що містить у собі дві треновані моделі для розпізнавання долоні: детектора долоні та модель розпізнавання, так званих, ключові точок високої точності. В свою чергу, розпізнавання жесту реалізовано за допомогою інструментів бібліотеки fingerpose. Вона дозволяє оброблювати ключові точки долоні, отримані за допомогою бібліотеки MediaPipe Handpose.

Програмний продукту розроблялся у редакторі коду Visual Studio Code, з використанням інструментів розробника браузера Google Chrome. Версіювання програмного продукту відбувалося за допомогою системи керування жестами GitHub.

Для підготовки навчальних матеріалів для користувача системи розпізнавання жестів для вивчення дактильної абетки було використано стандартні додатки iPhone, а само камеру та вбудований редактор фотографій.

4.2 Вимоги до технічного забезпечення

4.2.1 Загальні вимоги

Для коректної роботи системи розпізнавання жестів для вивчення дактильної абетки повинні виконуватися наступні вимоги до апаратного забезпечення:

а) процесор з тактовою частотою від 2.3 ГГц;

б) оперативна пам’ять від 8 ГБ.

Обов’язкова комп’ютерна периферія – камера.

Додаткові умови роботи системи розпізнавання жестів для вивчення дактильної абетки:

а) операційна система MacOS 11.0, Windows 10;

б) підключення до мережі інтеренет;

в) встановлений браузер Safari / Google Chrome / Mozilla Firefox;

г) для Windows, потрібно встановити програму Git bash.

4.3 Архітектура програмного забезпечення

4.3.1 Діаграма послідовності

Нижче на рисунку 4.2 описано функціювання системи розпізнавання жестів для вивчення дактильної абетки.

Рисунок 4.1 – Схема структурна послідовностей

4.3.2 Діаграма компонентів

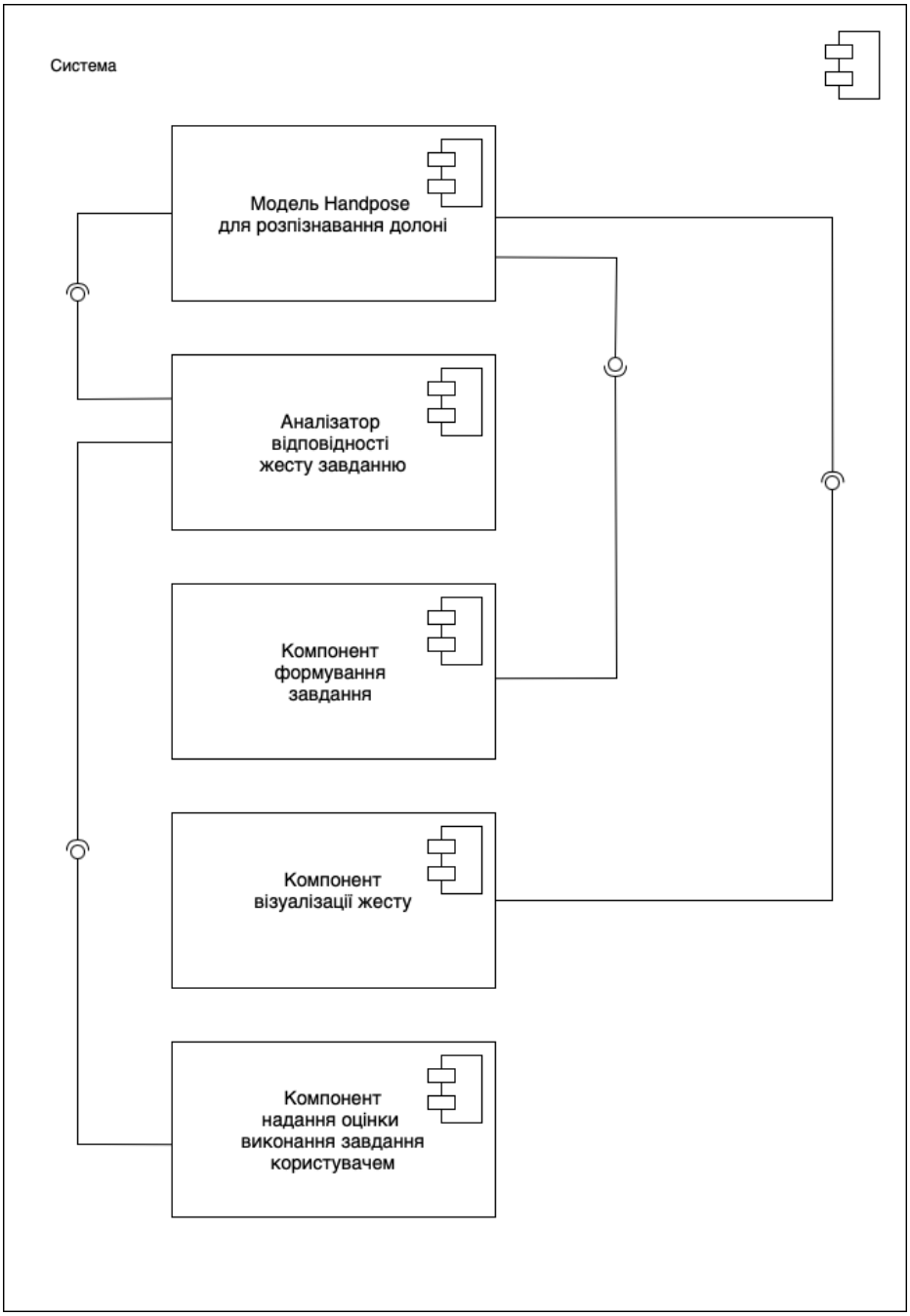


Рисунок 4.2 – Схема структурна компонентів

4.3.3 Специфікація функцій

Таблиця 4.1 – Основні функції системи

Функція	Опис
runHandpose()	Запуск та підтримання функціювання моделі HandPose за допомогою методу load(), виклик функції detect() кожні 100 мілісекунд.
detect()	Отримання і обробка відео-потоку та виклик усіх функцій для реалізації алгоритму розпізнавання жесту для провірки виконання завдання.
estimateHands()	Розпізнавання долоні та її ключових точок. Параметр функції – об’єкт відео-потоку. Повертає масив hand.
estimate()	Метод, який формує естімацію жесту на відповідність існуючим описаним жестам у системі.
addCurl()	Метод, який описує наявність і характер згину пальця. Використовується для опису жесту.
addDirection()	Метод, який описує напрям пальця. Використовується для опису жесту.
drawHand()	Утіліта для відображення ключових точок долоні на відео у реальному часу.

Висновок до розділу

У даному розділі було описано програмне та технічне забезпечення. Також були описані використані засоби розробки, сформовано вимоги до технічного забезпечення. Архітектура програмного забезпечення була описана за допомогою діаграми послідовностей та діаграми компонентів. Наведено перелік основних функцій системи.

5 ТЕХНОЛОГІЧНИЙ РОЗДІЛ

5.1 Керівництво користувача

Після запуску системи розпізнавання жестів для вивчення дактильної абетки у браузері користувач має можливість обрати зручний для нього спосіб навчання. На рисунку 5.1 зображено початковий екран веб-застосування.

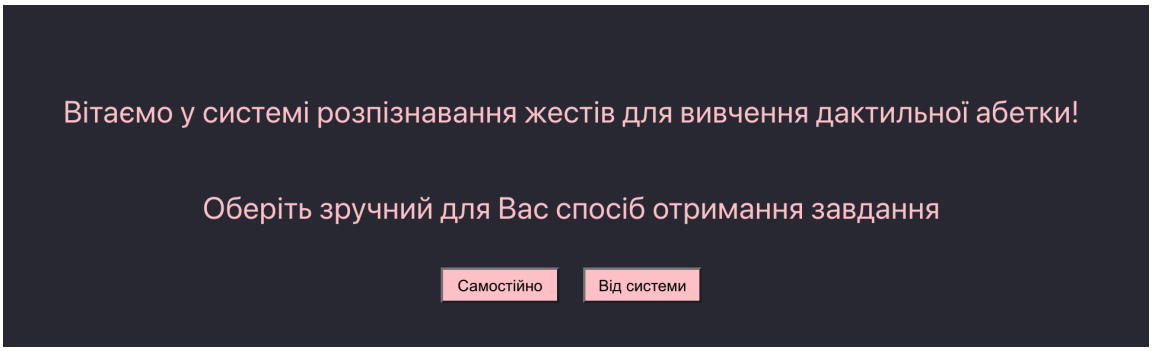


Рисунок 5.1 – Початковий екран

Якщо користувач обрав варіант «Самостійно», наступним кроком є вибір способу введення завдання. Можливі варіанти «Обрати зі списку» та «Ввести з клавіатури», які зображені на рисунку 5.2. Форми введення, що відповідають кожному з варіантів наведено на рисунку 5.3 та рисунку 5.4 відповідно.

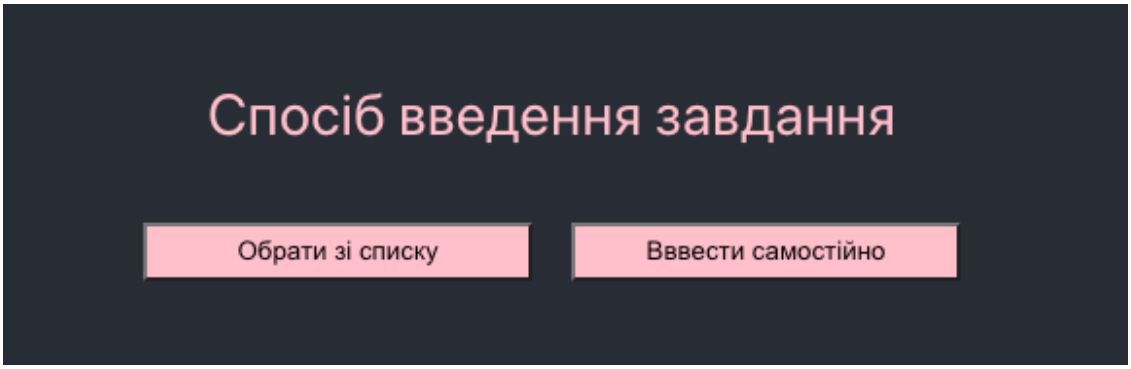


Рисунок 5.2 – Форма для вибору способу отримання літери завдання користувачем

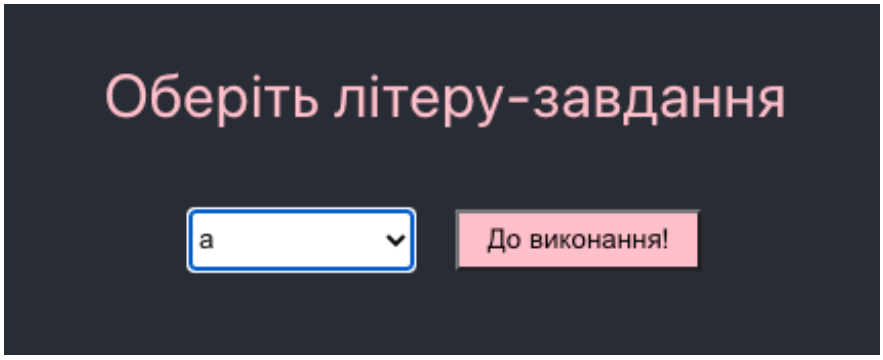


Рисунок 5.3 – Форма для обрання літери-завдання зі списку можливих

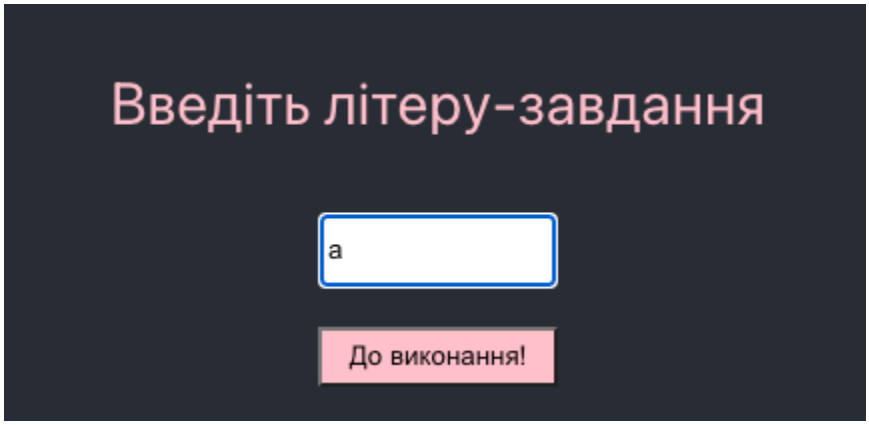


Рисунок 5.4 – Форма для введення літери-завдання з клавіатури

Якщо користувач обрав варіант отримання завдання «Від системи», наступним кроком є вибір типу завдання. Можливі варіанти «Одна рандомна літера» та «Усі літери поспіль», які зображені на рисунку 5.5.

Рисунок 5.5 – Форма для вибору типу завдання від системи

Після вибору типу завдання або отримання літери-завдання від користувача відкривається сторінка для контролю виконання завдання, яка зображена на рисунку 5.6.

На сторінці відображується відео з камери комп’ютера користувача, на якому присутня візуалізація жесту, який у поточний момент часу показує користувач. Також надається графічний приклад виконання поточного жесту-завдання.

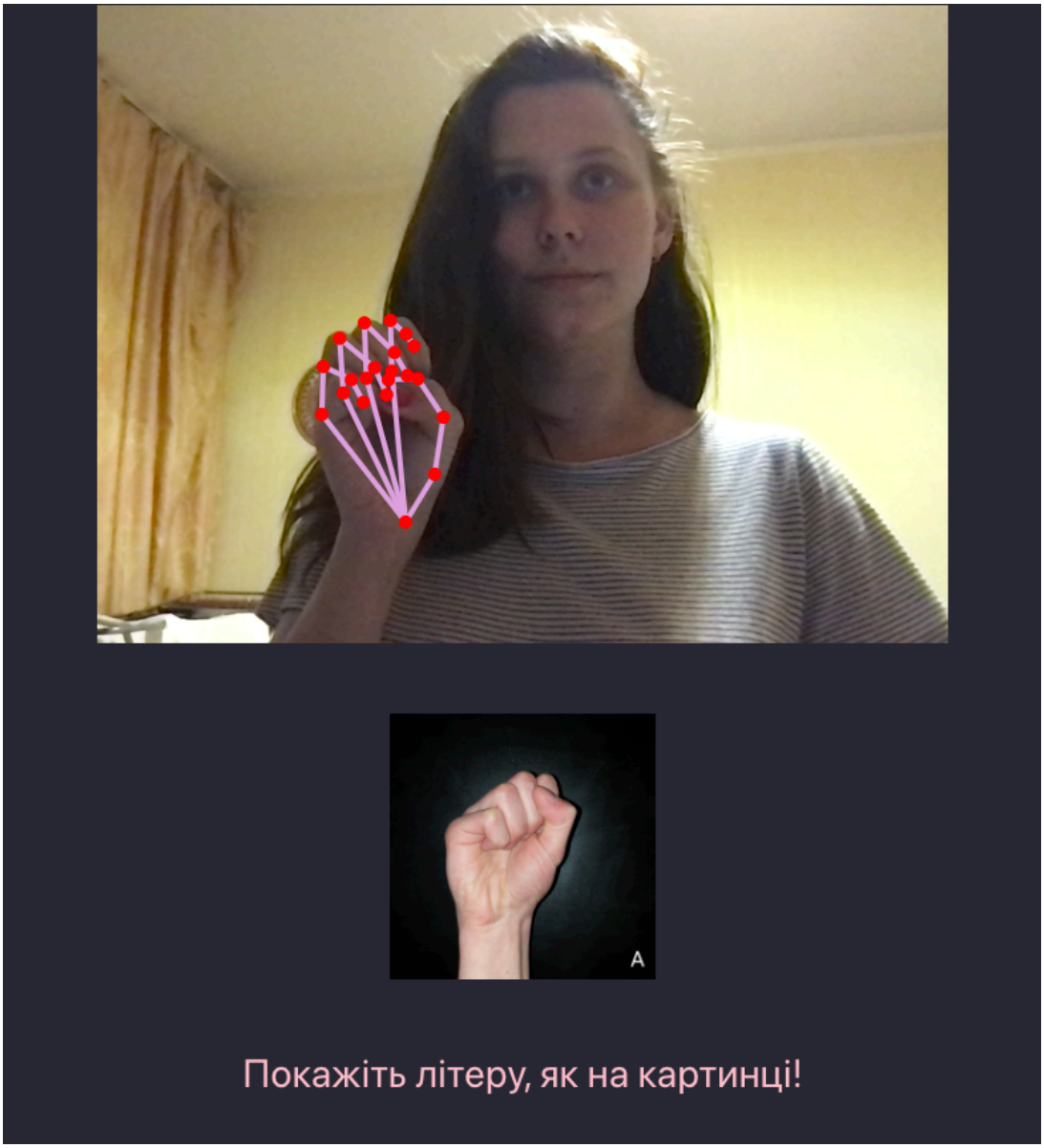


Рисунок 5.6 – Сторінка для контролю виконання завдання

Після виконання жесту, який відповідає поточній літері-завданню, користувач отримує сповіщення, що він успішно виконав завдання. Приклад сповіщення про виконання жесту-завдання «а» наведено на рисунку 5.7.

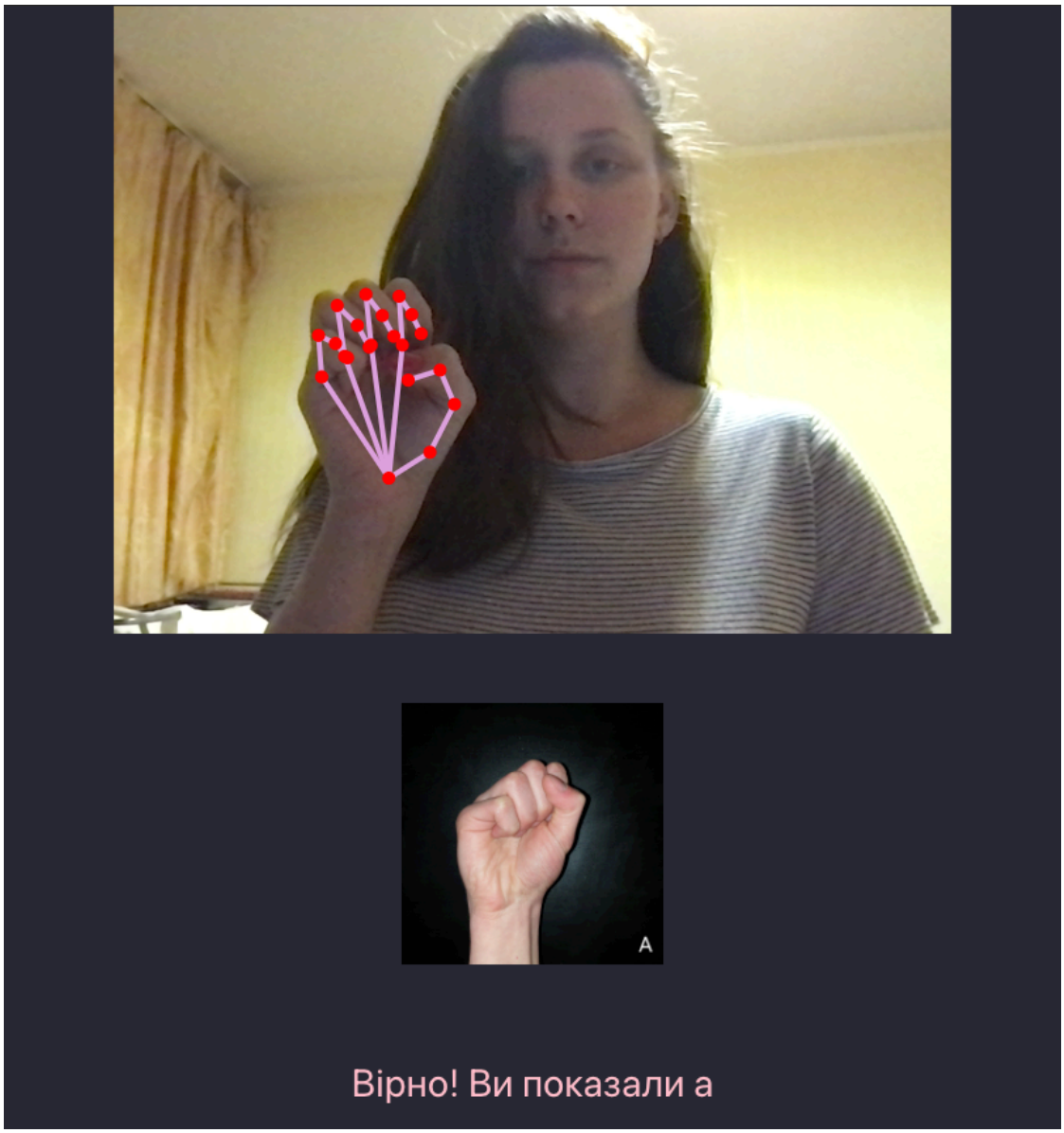


Рисунок 5.7 – Сповіщення про успішне виконання завдання

5.2 Випробування програмного продукту

5.2.1 Мета випробувань

Метою випробувань являється перевірка відповідності системи розпізнавання жестів для вивчення дактильної абетки вимогам технічного завдання. На рисунку 5.8 наведено діаграму діяльності, яка детально описує функціювання системи.

Рисунок 5.8 – Схема структурна діяльностіok

5.2.2 Загальні положення

Випробування проводяться на основі наступних документів:

– ГОСТ 34.603–92. Інформаційна технологія. Види випробувань автоматизованих систем;
– ГОСТ РД 50-34.698-90. Автоматизовані системи вимог до змісту документів.

5.2.3 Результати випробувань

Для підтвердження відповідності функціональним вимогам висунутим до системи розпізнавання жестів для вивчення дактильної абетки було проведено приймальне тестування. Усі функції були реалізовані відповідно до вимог, нижче наведено деякі проведені випробування та їх результати у таблицях 5.1, 5.2, 5.3, 5.4, 5.5, 5.6.

Таблиця 5.1 – Відтворення відео-потоку з камери комп’ютеру

Початкові умови	Відкрито веб-застосування «Система розпізнавання жестів для вивчення дактильної абетки».
Сценарій випробування	Обрати завдання із запропонованих варіантів, перейти до сторінки контролю виконання завдання, надати доступ на використання камери пристрою.
Очікуваний результат	Відео-потік запущено та він не зупиняється.
Отриманий результат	Відео-потік запущено, він не зупиняється.
Оцінка отриманого результату	Випробування успішне.

Таблиця 5.2 – Надання системою завдання користувачу

Початкові умови	Відкрито веб-застосування «Система розпізнавання жестів для вивчення дактильної абетки». Обрано варіант формування завдання системою.
Сценарій випробування	Обрати потрібний тип завдання («Одна рандомна літера» або «Усі літери поспіль»).
Очікуваний результат	Якщо обрано тип завдання «Одна рандомна літера», рандомна літера з переліку літер задана як завдання.
	Якщо обрано тип завдання «Усі літери поспіль», всі існуючі літери одна за одною задаються як завдання. Завдання змінюється на наступну літеру тільки після успішного виконання попереднього жесту завдання користувачем.

Продовження таблиці 5.2

Отриманий результат	Якщо обрано тип завдання «Одна рандомна літера», рандомна літера з переліку літер задана як завдання.
	Якщо обрано тип завдання «Усі літери поспіль», всі існуючі літери одна за одною задаються як завдання. Завдання змінюється на наступну літеру тільки після успішного виконання попереднього жесту завдання користувачем.
Оцінка отриманого результату	Випробування успішне.

Таблиця 5.3 – Можливість формування завдання користувачем

Початкові умови	Відкрито веб-застосування «Система розпізнавання жестів для вивчення дактильної абетки».
Сценарій випробування	Обрати варіант отримання завдання «Від системи», обрати потрібний тип завдання («Одна рандомна літера» або «Усі літери поспіль»).
Очікуваний результат	Якщо обрано тип завдання «Одна рандомна літера», рандомна літера з переліку літер задана як завдання.
	Якщо обрано тип завдання «Усі літери поспіль», всі існуючі літери одна за одною задаються як завдання.
Отриманий результат	Якщо обрано тип завдання «Одна рандомна літера», рандомна літера з переліку літер задана як завдання.
	Якщо обрано тип завдання «Одна рандомна літера», рандомна літера з переліку літер задана як завдання.

Таблиця 5.4 – Візуалізація жесту на відео

Початкові умови	Відкрито веб-застосування, надано дозвіл на використання камери у браузері. Обрано один з форматів отримання завдання, відкрито сторінку контролю виконання.
Сценарій випробування	Користувач, знаходячись перед камерою, відтворює жест-завдання.
Очікуваний результат	На відео на місці долоні відображується «скелет» долоні, візуалізація не зупиняється, доки рука є на відео.
Отриманий результат	На відео на місці долоні відображується «скелет» долоні, візуалізація не зупиняється, доки рука є на відео.
Оцінка отриманого результату	Випробування успішне.

Таблиця 5.5 – Відображення учбового матеріалу у відповідності до завдання

Початкові умови	Відкрито веб-застосування, обрано один з форматів отримання завдання, відкрито сторінку контролю виконання.
Сценарій випробування	Користувач отримує приклад виконання завдання у вигляді картинки.
Очікуваний результат	На картинці зображено жест, що відповідає жесту-завданнню.
Отриманий результат	На картинці зображено жест, що відповідає жесту-завданнню.
Оцінка отриманого результату	Випробування успішне.

Таблиця 5.6 – Надання системою оцінки виконання жесту користувачу

Початкові умови	Відкрито веб-застосування, надано дозвіл на використання камери у браузері. Обрано один з форматів отримання завдання, відкрито сторінку контролю виконання.
Сценарій випробування	Користувач, знаходячись перед камерою, відтворює жест-завдання як показано на графічному зображені.
Очікуваний результат	Відображується повідомлення про успішне виконання завдання. Після чого відкривається сторінка вибору способу отримання завдання, якщо було завдання било показати одну літеру. У випадку, коли завдання показати усі літери поспіль, наступне завдання відображається.
Отриманий результат	Відображується повідомлення про успішне виконання завдання. Після чого відкривається сторінка вибору способу отримання завдання, якщо було завдання било показати одну літеру. У випадку, коли завдання показати усі літери поспіль, наступне завдання відображається.
Оцінка отриманого результату	Випробування успішне.

Висновок до розділу

В даному розділі було надано керівництво користувача, яке описує взаємодію користувача із системою розпізнавання жестів для вивчення дактильної абетки. Були проведені випробовування, а саме приймальне тестування, які довели відповідність веб-застосування вимогам.

ЗАГАЛЬНІ ВИСНОВКИ

Впродовж роботи над дипломним проєктом було вдосконалено практичні та теоретичні знання у системному проектуванні. Було розроблено веб-застосування «Система розпізнавання жестів для вивчення дактильної абетки».

На початку роботи було досліджено предметне середовище, сформовані підстави розробки системи та визначені можливості користувача при роботі із застосунком. Одним з перших кроків у роботі над дипломним проєктом став аналіз найпопулярніших аналогів, які допомагають користувачу у вивченні дактильної абетки. На підставі даного аналізу було прийняте рішення щодо розробки системи та сформовано її ключову особливість - оцінювання виконання завдання (жесту). Було сформульовано функціональні вимоги, яким має відповідати система розпізнавання жестів для вивчення дактильної абетки.

Було описано вхідні та вихідні дані системи, наведені приклади вхідних даних (відео-потік). Також було продемонстровано вихідні данні , а саме оцінка виконання завдання (жесту) користувачем. Додатково визначено структуру масивів інформації, яка використовується для розпізнавання жестів.

Була сформульована змістовна постановка задачі та математична постановки задачі. Впродовж роботи над розділом з математичного забезпечення було досліджено та описано основні існуючи методи розв’язання подібних задач. Базуючись на зібраних даних било прийнято рішення використовувати TensorFlow.js для реалізації методу розв’язання.

Паралельно з розробкою веб-застосування було описано програмне та технічне забезпечення. У четвертому розділі були описані використані засоби розробки, сформовано вимоги до технічного забезпечення. Архітектура програмного забезпечення була описана за допомогою діаграми

послідовностей та діаграми компонентів, наведено перелік основних функцій системи.

Останній розділ присвячено керівництву користувача, яке описує взаємодію користувача із системою розпізнавання жестів для вивчення дактильної абетки. Також після проведення випробовуваннь, а саме приймального тестування, було встановлено відповідність веб-застосування висунутим вимогам.

Наступними кроками на шляху розвитку системи розпізнавання жестів для вивчення дактильної абетки можуть бути покращення користувацького інтерфейсу і розширення бібліотеки жестів.

ПЕРЕЛІК ПОСИЛАНЬ

1) Руденко А.О. Система розпізнавання жестів для вивчення дактильної абетки //V Всеукраїнська науково- практична конференція молодих вчених та студентів « Інформатика та обчислювальна техніка-IOT-2021» Секція кафедри автоматизованих систем обробки інформації і управління. Матеріали конференції. – 2021. –С.

2) Face and hand tracking in the browser with MediaPipe and TensorFlow.js. URL: https://blog.tensorflow.org/2020/03/face-and-hand-tracking-in-browser-with-mediapipe-and-tensorflowjs.html

3) On-Device, Real-Time Hand Tracking with MediaPipe. URL: https://ai.googleblog.com/2019/08/on-device-real-time-hand-tracking-with.html

4) Обзор Tensorflow.JS: машинное обучение на JavaScript. URL: https://neurohive.io/ru/tutorial/obzor-tensorflow-js-mashinnoe-obuchenie-na-javascript/

5) React. Getting started. URL: https://reactjs.org/docs/getting-started.html

Додаток А

Тексти програмного коду

Система розпізнавання жестів для вивчення дактильної абетки

(Найменування програми (документа))

DVD-R

(Вид носія даних)

20 арк, 503 Мб

(Обсяг програми (документа) , арк., Мб)

Київ – 2021 року

Index.js

```
import React from 'react';
import ReactDOM from 'react-dom';
import './index.css';
import App from './App';
import reportWebVitals from './reportWebVitals';

ReactDOM.render(
  <React.StrictMode>
    <App />
  </React.StrictMode>,
  document.getElementById('root')
);

reportWebVitals();
```

Index.css

```
body {
  margin: 0;
  font-family: -apple-system, BlinkMacSystemFont, 'Segoe UI', 'Roboto', 'Oxygen',
    'Ubuntu', 'Cantarell', 'Fira Sans', 'Droid Sans', 'Helvetica Neue',
    sans-serif;
  -webkit-font-smoothing: antialiased;
  -moz-osx-font-smoothing: grayscale;
}

code {
  font-family: source-code-pro, Menlo, Monaco, Consolas, 'Courier New',
    monospace;
}
```

App.js

```
import React, {useRef, useState, useEffect} from 'react';
import * as tf from "@tensorflow/tfjs";
import * as handpose from "@tensorflow-models/handpose";
import Webcam from "react-webcam";
import './App.css';
// import { scryRenderedComponentsWithType } from 'react-dom/cjs/react-dom-test-
utils.development'; // ?
import { drawHand } from "./utilities";
import * as fp from 'fingerpose';

import {aSign1} from './letters/A_1';
import {bSign2} from './letters/B_2';
import {vSign3} from './letters/V_3';
```

```
import {gSign4} from './letters/G_4';
import {gaSign5} from './letters/Ga_5';
import {eSign7} from './letters/E_7';
import {dgSign9} from './letters/Dg_9';
import {zSign10} from './letters/Z_10';
import {iSign12} from './letters/I_12';
import {nSign18} from './letters/N_18';
import {oSign19} from './letters/O_19';
import {pSign20} from './letters/P_20';
// import {rSign21} from './letters/R_21';
import {sSign22} from './letters/S_22';
// import {tSign23} from './letters/T_23';
// import {shSign29} from './letters/Sh_29';
// import {softSign33} from './letters/Soft_33';



import thumbs_up from "./thumbs_up.png";
import a_example from "./letter_examples/aSign1.png";
import b_example from "./letter_examples/bSign2.png";
import v_example from "./letter_examples/vSign3.png";
import g_example from "./letter_examples/gSign4.png";
import ga_example from "./letter_examples/gaSign5.png";


import { Reduction, Sign } from '@tensorflow/tfjs';


function App() {

  const webcamRef = useRef(null);
  const canvasRef = useRef(null);



  const [sign, setSign] = useState("");
  const [taskSign, setTaskSign] = useState("");
  const [done, setDone] = useState(false);
  const [signNumber, setSignNumber] = useState(0);
  let numState = 0;
  const [example, setExample] = useState(true);



  const images = {
    // 0: thumbs_up,
    0: a_example,
    1: b_example,
    2: v_example,
    3: g_example,
    4: ga_example
  };
```

```
const runHandpose = async () => {
    debugger;
    const net = await handpose.load();
    // console.log('Handpose model is loaded.');

    console.log('Модель завантажена.');

    // continuous detection in a loop
    setInterval(()=>{
      detect(net);
    }, 100);
  };

  const detect = async (net) => {

    // validation of dtoIn
    if (typeof webcamRef.current !== "undefined" && webcamRef.current !== null &&
webcamRef.current.video.readyState === 4) {

      // get video properties
      const video = webcamRef.current.video;
      // video.style.transform = 'scale(-1, 1)';
      const videoWidth = webcamRef.current.video.videoWidth;
      const videoHeight = webcamRef.current.video.videoHeight;
      // set video height and width
      webcamRef.current.video.width = videoWidth;
      webcamRef.current.video.height = videoHeight;
      // set canvas height and width
      canvasRef.current.width = videoWidth;
      canvasRef.current.height =videoHeight;

      // make detections
      const hand = await net.estimateHands(video);

      // continuos gesture detections
      if (hand.length > 0) {
        setDone(false);

        debugger;

        const GE = new fp.GestureEstimator([
          // fp.Gestures.ThumbsUpGesture,
          aSign1,
          bSign2,
          vSign3,
          gSign4,
          gaSign5,
          eSign7,
          dgSign9,
          zSign10,
```

```
          iSign12,
          nSign18,
    oSign19,
          pSign20,
          // rSign21
          sSign22
          // tSign23,
          // shSign29,
          // softSign33
        ]);


        // get only gestures with 8+ confidance
        const gesture = await GE.estimate(hand[0].landmarks, 8);



        // for all possible gestures
        if (gesture.gestures !== undefined && gesture.gestures.length > 0) {
          console.log(gesture.gestures);
          const confidence = gesture.gestures.map(
            (prediction) => prediction.confidence
          );
          const maxConfidence = confidence.indexOf(
            Math.max.apply(null, confidence)
          );


          // just detected gesture
          setSign(gesture.gestures[maxConfidence].name);
          console.log("now gesture: " + gesture.gestures[maxConfidence].name);

          // setTaskSign(GE.gestures[signNumber].name);
          console.log("your task : " + GE.gestures[numState].name);

          setExample(true)



          // user accomplished the task
          if (gesture.gestures[maxConfidence].name == GE.gestures[numState].name) {
            console.log("Gesture в detected!");

            setDone(true);

            numState = numState + 1;
            setSignNumber(prevNumber => prevNumber + 1);


            // setDone(false);
            setExample(false);
```

```
            debugger;

            // setTimeout(()=>{
            //   setDone(false);
            //   setExample(false);
            // }, 3000);

          }
        }

      }

      // draw mesh
      const ctx = canvasRef.current.getContext("2d");
      drawHand(hand, ctx);
    }
  };

  // runHandpose();
  useEffect(()=>{runHandpose()},[]);

  return (
    <div className="App">
      <header className="App-header">

        <Webcam
          ref={webcamRef}
          style={{
            position:"absolute",
            marginLeft:"auto",
            marginRight:"auto",
            top: 0,
            left:0,
            right:0,
            textAlign:"center",
            zIndex:8,
            width: 640,
            height: 480,
          }}
        />
        <canvas
          ref={canvasRef}
          style={{
            position:"absolute",
            marginLeft:"auto",
            marginRight:"auto",
            top: 0,
            left:0,
            right:0,
            textAlign:"center",
            zIndex:9,
            width: 640,
```

```
            height: 480,
          }}
  />

        {example == true ?
          <img src={images[signNumber]} style={{
            position: "absolute",
            top: '55vh',
            bottom: '15vh',
            marginLeft: "auto",
            marginRight: "auto",
            textAlign: "center",
            zIndex: 10,
            width: 200,
            height: 200,
            }} /> : ""}

        {done !== true ?
          <div style={{
            position: "absolute",
            bottom: 10,
            marginLeft: "auto",
            marginRight: "auto",
            textAlign: "center",
            zIndex: 10,
            width: '70vh',
            height: '10vw',
            padding: 10,
        }}>
          <p style={{color:'pink'}}>Покажіть літеру, як на картинці!</p>
          {/* Номер картинки {signNumber}. */}
        </div> : ""}

        {done == true ?
          <div style={{
            position: "absolute",
            bottom: 0,
            marginLeft: "auto",
            marginRight: "auto",
            textAlign: "center",
            zIndex: 10,
            width: '70vh',
            height: '10vw',
        }}>
          <p style={{color:'pink'}}>Покажіть літеру, як на картинці!</p>

          {/* Вірно! Ви показали а {sign}! */}
        </div> : ""}

            {/* <div style={{
            position: "absolute",
            bottom: 0,
```

```
            marginLeft: "auto",
            marginRight: "auto",
textAlign: "center",
            zIndex: 10,
            width: '70vh',
            height: '10vw',
          }}>
           <button onClick={() => setNumber(number+1)}>Наступна літера</button>
          </div> */}

          {/* <input style={{
            position: "absolute",
            bottom: 15,
            marginLeft: "auto",
            marginRight: "auto",
            textAlign: "center",
            zIndex: 10,
            width: '70vh',
            height: '10vw',
            padding: 10,
          }}>
          </input> */}


      </header>
    </div>
  );
}

export default App;
```

App.css

```
.App {
  text-align: center;
}

.App-logo {
  height: 40vmin;
  pointer-events: none;
}

@media (prefers-reduced-motion: no-preference) {
  .App-logo {
    animation: App-logo-spin infinite 20s linear;
  }
}

.App-header {
  background-color: #282c34;
  min-height: 100vh;
```

```
  display: flex;
  flex-direction: column;
 align-items: center;
  justify-content: center;
  font-size: calc(10px + 2vmin);
  color: white;
 }

 .App-link {
  color: #61dafb;
 }

 @keyframes App-logo-spin {
  from {
    transform: rotate(0deg);
  }
  to {
    transform: rotate(360deg);
  }
 }
```

Utillities.js

```
// points of finger
const fingerJoints = {
    thumb: [0,1, 2, 3, 4],
    indexFinger: [0, 5, 6, 7, 8],
    middleFinger: [0, 9, 10, 11, 12],
    ringFinger: [0, 13, 14, 15, 16],
    pinky: [0, 17, 18, 19, 20],
};

//  drawing function
export const drawHand = (predictions, ctx) => {
    // check if we have predictions
    if (predictions.length > 0) {
        predictions.forEach((prediction) => {
            // recive landmarks
            const landmarks = prediction.landmarks;

            // loop through fingers
            for (let j = 0; j < Object.keys(fingerJoints).length; j++) {
                let finger = Object.keys(fingerJoints)[j];
                // loop through pairs of joints
                for (let k = 0; k < fingerJoints[finger].length - 1; k++) {
                    // gets pair of joints
                    const firstJointIndex = fingerJoints[finger][k];
                    const secondJointIndex = fingerJoints[finger][k+1];
                    // draw link
                    ctx.beginPath();
                    ctx.moveTo(
```

```
                    landmarks[firstJointIndex][0],
                    landmarks[firstJointIndex][1]
);
                ctx.lineTo(
                    landmarks[secondJointIndex][0],
                    landmarks[secondJointIndex][1]
                );
                ctx.strokeStyle = "plum";
                ctx.lineWidth = 4;
                ctx.stroke();
            }
        }

        // loop and draw
        for (let i = 0; i < landmarks.length; i++) {
            // get x
            const x = landmarks[i][0];
            // get y
            const y = landmarks[i][1];
            // drawign
            ctx.beginPath();
            ctx.arc(x, y, 5, 0, 3 * Math.PI);
            ctx.fillStyle = "red";
            ctx.fill();
        }
    });
  };
};
```

A_1.js

```
// import dependencies
import {Finger, FingerCurl, FingerDirection, GestureDescription} from 'fingerpose';

// define gesture description
export const aSign1 =  new GestureDescription('a');

// thumb
aSign1.addCurl(Finger.Thumb, FingerCurl.HalfCurl, 0.75)
aSign1.addDirection(Finger.Thumb, FingerDirection.DiagonalUpLeft , 0.75)
aSign1.addDirection(Finger.Thumb, FingerDirection.DiagonalUpRight, 0.75)

// index
aSign1.addCurl(Finger.Index, FingerCurl.FullCurl, 1.0)
aSign1.addCurl(Finger.Index, FingerCurl.HalfCurl, 0.75)
aSign1.addDirection(Finger.Index, FingerDirection.DiagonalUpLeft, 0.75)
aSign1.addDirection(Finger.Index, FingerDirection.DiagonalUpRight, 0.75)

// middle
aSign1.addCurl(Finger.Middle, FingerCurl.FullCurl, 1.0)
aSign1.addCurl(Finger.Middle, FingerCurl.HalfCurl, 0.75)
```

```
aSign1.addDirection(Finger.Middle, FingerDirection.DiagonalUpLeft, 0.75)
aSign1.addDirection(Finger.Middle, FingerDirection.DiagonalUpRight, 0.75)
// ring
aSign1.addCurl(Finger.Ring, FingerCurl.FullCurl, 1.0)
aSign1.addCurl(Finger.Ring, FingerCurl.HalfCurl, 0.75)
aSign1.addDirection(Finger.Ring, FingerDirection.DiagonalUpLeft, 0.75)
aSign1.addDirection(Finger.Ring, FingerDirection.DiagonalUpRight, 0.75)

// pinky
aSign1.addCurl(Finger.Pinky, FingerCurl.FullCurl, 1.0)
aSign1.addCurl(Finger.Pinky, FingerCurl.HalfCurl, 0.75)
aSign1.addDirection(Finger.Pinky, FingerDirection.DiagonalUpLeft, 0.75)
aSign1.addDirection(Finger.Pinky, FingerDirection.DiagonalUpRight, 0.75)
```

B_2.js

```
// import dependencies
import {Finger, FingerCurl, FingerDirection, GestureDescription} from 'fingerpose';

// define gesture description
export const bSign2 =  new GestureDescription('б');

// thumb
bSign2.addCurl(Finger.Thumb, FingerCurl.HalfCurl, 1.0)
bSign2.addDirection(Finger.Thumb, FingerDirection.DiagonalUpLeft , 1.0)
bSign2.addDirection(Finger.Thumb, FingerDirection.DiagonalUpRight, 1.0)

// index
bSign2.addCurl(Finger.Index, FingerCurl.NoCurl, 1.0)
bSign2.addDirection(Finger.Index, FingerDirection.VerticalUp, 1.0)

// middle
bSign2.addCurl(Finger.Middle, FingerCurl.FullCurl, 1.0)
bSign2.addCurl(Finger.Middle, FingerCurl.HalfCurl, 0.75)
bSign2.addDirection(Finger.Middle, FingerDirection.VerticalUp, 1.0)
bSign2.addDirection(Finger.Middle, FingerDirection.DiagonalUpLeft, 0.75)
bSign2.addDirection(Finger.Middle, FingerDirection.DiagonalUpRight, 0.75)

// ring
bSign2.addCurl(Finger.Ring, FingerCurl.FullCurl, 1.0)
bSign2.addDirection(Finger.Ring, FingerDirection.VerticalUp, 1.0)

// pinky
bSign2.addCurl(Finger.Pinky, FingerCurl.FullCurl, 1.0)
bSign2.addDirection(Finger.Pinky, FingerDirection.DiagonalUpLeft, 1.0)
bSign2.addDirection(Finger.Pinky, FingerDirection.DiagonalUpRight, 1.0)
```

V_3.js

```
// import dependencies
import {Finger, FingerCurl, FingerDirection, GestureDescription} from 'fingerpose';
```

```
// define gesture description
export const vSign3 =  new GestureDescription('в');

// thumb
vSign3.addCurl(Finger.Thumb, FingerCurl.NoCurl, 1.0);
vSign3.addDirection(Finger.Thumb, FingerDirection.VerticalUp, 1.0);

// index
vSign3.addCurl(Finger.Index, FingerCurl.NoCurl, 1.0);
vSign3.addDirection(Finger.Index, FingerDirection.VerticalUp, 1.0);

// middle
vSign3.addCurl(Finger.Middle, FingerCurl.NoCurl, 1.0);
vSign3.addDirection(Finger.Middle, FingerDirection.VerticalUp, 1.0);

// ring
vSign3.addCurl(Finger.Ring, FingerCurl.NoCurl, 1.0);
vSign3.addCurl(Finger.Ring, FingerCurl.HalfCurl, 0.0);
vSign3.addDirection(Finger.Ring, FingerDirection.VerticalUp, 1.0);

// pinky
vSign3.addCurl(Finger.Pinky, FingerCurl.NoCurl, 1.0);
vSign3.addDirection(Finger.Pinky, FingerDirection.VerticalUp, 1.0);
```

G_4.js

```
// import dependencies
import {Finger, FingerCurl, FingerDirection, GestureDescription} from 'fingerpose';

// define gesture description
export const gSign4 =  new GestureDescription('г');

// thumb
gSign4.addCurl(Finger.Thumb, FingerCurl.NoCurl, 1.0);
gSign4.addDirection(Finger.Thumb, FingerDirection.HorizontalLeft, 1.0);
gSign4.addDirection(Finger.Thumb, FingerDirection.HorizontalRight, 1.0);

// index
gSign4.addCurl(Finger.Index, FingerCurl.NoCurl, 1.0);
gSign4.addDirection(Finger.Index, FingerDirection.VerticalDown, 1.0);

// middle
gSign4.addCurl(Finger.Middle, FingerCurl.FullCurl, 1.0)
gSign4.addDirection(Finger.Middle, FingerDirection.VerticalDown, 1.0)

// ring
gSign4.addCurl(Finger.Ring, FingerCurl.FullCurl, 1.0)
gSign4.addDirection(Finger.Ring, FingerDirection.VerticalDown, 1.0)

// pinky
gSign4.addCurl(Finger.Pinky, FingerCurl.FullCurl, 1.0)
```

```
gSign4.addDirection(Finger.Pinky, FingerDirection.VerticalDown, 1.0)
```

Ga_5.js

```
// import dependencies
import {Finger, FingerCurl, FingerDirection, GestureDescription} from 'fingerpose';

// define gesture description
export const gaSign5 =  new GestureDescription('r');

// thumb
gaSign5.addCurl(Finger.Thumb, FingerCurl.FullCurl, 1.0);
gaSign5.addCurl(Finger.Thumb, FingerCurl.HalfCurl, 0.75);
gaSign5.addDirection(Finger.Thumb, FingerDirection.DiagonalDownLeft, 1.0);
gaSign5.addDirection(Finger.Thumb, FingerDirection.DiagonalDownLeftt, 1.0);

// index
gaSign5.addCurl(Finger.Index, FingerCurl.NoCurl, 1.0);

gaSign5.addDirection(Finger.Index, FingerDirection.VerticalDown, 1.0);

// middle
gaSign5.addCurl(Finger.Middle, FingerCurl.FullCurl, 1.0)
gaSign5.addDirection(Finger.Middle, FingerDirection.VerticalDown, 1.0)

// ring
gaSign5.addCurl(Finger.Ring, FingerCurl.FullCurl, 1.0)
gaSign5.addDirection(Finger.Ring, FingerDirection.VerticalDown, 1.0)

// pinky
gaSign5.addCurl(Finger.Pinky, FingerCurl.FullCurl, 1.0)
gaSign5.addDirection(Finger.Pinky, FingerDirection.VerticalDown, 1.0)
```

E_7.js

```
// import dependencies
import {Finger, FingerCurl, FingerDirection, GestureDescription} from 'fingerpose';

// define gesture description
export const eSign7 =  new GestureDescription('e');

// thumb
eSign7.addCurl(Finger.Thumb, FingerCurl.NoCurl, 1.0);
eSign7.addCurl(Finger.Thumb, FingerCurl.HalfCurl, 0.5);
eSign7.addDirection(Finger.Thumb, FingerDirection.HorizontalLeft, 1.0);
eSign7.addDirection(Finger.Thumb, FingerDirection.HorizontalRight, 1.0);
eSign7.addDirection(Finger.Thumb, FingerDirection.DiagonalUpLeft, 0.5);
eSign7.addDirection(Finger.Thumb, FingerDirection.DiagonalUpRight, 0.5);

// index
eSign7.addCurl(Finger.Index, FingerCurl.HalfCurl, 1.0);
```

```
eSign7.addCurl(Finger.Index, FingerCurl.FullCurl, 0.75);
eSign7.addDirection(Finger.Index, FingerDirection.HorizontalLeft, 1.0);
eSign7.addDirection(Finger.Index, FingerDirection.HorizontalRight, 1.0);
eSign7.addDirection(Finger.Index, FingerDirection.DiagonalUpLeft, 0.75);
eSign7.addDirection(Finger.Index, FingerDirection.DiagonalUpRight, 0.75)

// middle
eSign7.addCurl(Finger.Middle, FingerCurl.HalfCurl, 1.0);
eSign7.addCurl(Finger.Middle, FingerCurl.FullCurl, 0.75);
eSign7.addDirection(Finger.Middle, FingerDirection.HorizontalLeft, 1.0);
eSign7.addDirection(Finger.Middle, FingerDirection.HorizontalRight, 1.0);
eSign7.addDirection(Finger.Middle, FingerDirection.DiagonalUpLeft, 0.75);
eSign7.addDirection(Finger.Middle, FingerDirection.DiagonalUpRight, 0.75)

// ring
eSign7.addCurl(Finger.Ring, FingerCurl.HalfCurl, 1.0);
eSign7.addCurl(Finger.Ring, FingerCurl.FullCurl, 0.75);
eSign7.addDirection(Finger.Ring, FingerDirection.HorizontalLeft, 1.0);
eSign7.addDirection(Finger.Ring, FingerDirection.HorizontalRight, 1.0);
eSign7.addDirection(Finger.Ring, FingerDirection.DiagonalUpLeft, 0.75);
eSign7.addDirection(Finger.Ring, FingerDirection.DiagonalUpRight, 0.75)

// pinky
eSign7.addCurl(Finger.Pinky, FingerCurl.HalfCurl, 1.0);
eSign7.addCurl(Finger.Pinky, FingerCurl.FullCurl, 0.75);
eSign7.addDirection(Finger.Pinky, FingerDirection.HorizontalLeft, 1.0);
eSign7.addDirection(Finger.Pinky, FingerDirection.HorizontalRight, 1.0);
eSign7.addDirection(Finger.Pinky, FingerDirection.DiagonalUpLeft, 0.75);
eSign7.addDirection(Finger.Pinky, FingerDirection.DiagonalUpRight, 0.75)
```

Dg_9.js

```
// import dependencies
import {Finger, FingerCurl, FingerDirection, GestureDescription} from 'fingerpose';

// define gesture description
export const dgSign9 =  new GestureDescription('ж');

// thumb
dgSign9.addCurl(Finger.Thumb, FingerCurl.NoCurl, 1.0);
dgSign9.addDirection(Finger.Thumb, FingerDirection.HorizontalLeft, 1.0);
dgSign9.addDirection(Finger.Thumb, FingerDirection.HorizontalRight, 1.0);

// index
dgSign9.addCurl(Finger.Index, FingerCurl.NoCurl, 1.0);
dgSign9.addDirection(Finger.Index, FingerDirection.HorizontalLeft, 1.0);
dgSign9.addDirection(Finger.Index, FingerDirection.HorizontalRight, 1.0);

// middle
dgSign9.addCurl(Finger.Middle, FingerCurl.NoCurl, 1.0);
dgSign9.addDirection(Finger.Middle, FingerDirection.HorizontalLeft, 1.0);
dgSign9.addDirection(Finger.Middle, FingerDirection.HorizontalRight, 1.0);
```

```
// ring
dgSign9.addCurl(Finger.Ring, FingerCurl.NoCurl, 1.0);
dgSign9.addDirection(Finger.Ring, FingerDirection.HorizontalLeft, 1.0);
dgSign9.addDirection(Finger.Ring, FingerDirection.HorizontalRight, 1.0);

// pinky
dgSign9.addCurl(Finger.Pinky, FingerCurl.NoCurl, 1.0);
dgSign9.addDirection(Finger.Pinky, FingerDirection.HorizontalLeft, 1.0);
dgSign9.addDirection(Finger.Pinky, FingerDirection.HorizontalRight, 1.0);
```

Z_10.js

```
// import dependencies
import {Finger, FingerCurl, FingerDirection, GestureDescription} from 'fingerpose';

// define gesture description
export const zSign10 =  new GestureDescription('з');

// thumb
zSign10.addCurl(Finger.Thumb, FingerCurl.NoCurl, 1.0);
zSign10.addDirection(Finger.Thumb, FingerDirection.DiagonalUpLeft, 1.0);
zSign10.addDirection(Finger.Thumb, FingerDirection.DiagonalUpRight, 1.0);

// index
zSign10.addCurl(Finger.Index, FingerCurl.NoCurl, 1.0);
zSign10.addDirection(Finger.Index, FingerDirection.DiagonalUpLeft, 1.0);
zSign10.addDirection(Finger.Index, FingerDirection.DiagonalUpRight, 1.0);

//middle
zSign10.addCurl(Finger.Middle, FingerCurl.FullCurl, 1.0);
zSign10.addDirection(Finger.Middle, FingerDirection.DiagonalUpLeft, 1.0);
zSign10.addDirection(Finger.Middle, FingerDirection.DiagonalUpRight, 1.0);

// ring
zSign10.addCurl(Finger.Ring, FingerCurl.FullCurl, 1.0);
zSign10.addDirection(Finger.Ring, FingerDirection.HorizontalLeft, 1.0);
zSign10.addDirection(Finger.Ring, FingerDirection.HorizontalRight, 1.0);

// pinky
zSign10.addCurl(Finger.Pinky, FingerCurl.FullCurl, 1.0);
zSign10.addCurl(Finger.Pinky, FingerCurl.HalfCurl, 0.75);
zSign10.addDirection(Finger.Pinky, FingerDirection.HorizontalLeft, 1.0);
zSign10.addDirection(Finger.Pinky, FingerDirection.HorizontalRight, 1.0);
```

I_12.js

```
// import dependencies
import {Finger, FingerCurl, FingerDirection, GestureDescription} from 'fingerpose';

// define gesture description
export const iSign12 =  new GestureDescription('і');
```

```
// thumb
iSign12.addCurl(Finger.Thumb, FingerCurl.HalfCurl, 1.0);
iSign12.addDirection(Finger.Thumb, FingerDirection.DiagonalUpLeft, 1.0);
iSign12.addDirection(Finger.Thumb, FingerDirection.DiagonalUpRight, 1.0);

// index
iSign12.addCurl(Finger.Index, FingerCurl.FullCurl, 1.0);
iSign12.addDirection(Finger.Index, FingerDirection.DiagonalUpLeft, 1.0);
iSign12.addDirection(Finger.Index, FingerDirection.DiagonalUpRight, 1.0);

// middle
iSign12.addCurl(Finger.Middle, FingerCurl.FullCurl, 1.0);
iSign12.addDirection(Finger.Middle, FingerDirection.DiagonalUpLeft, 1.0);
iSign12.addDirection(Finger.Middle, FingerDirection.DiagonalUpRight, 1.0);

// ring
iSign12.addCurl(Finger.Ring, FingerCurl.FullCurl, 1.0);
// iSign12.addDirection(Finger.Ring, FingerDirection.VerticalUp, 1.0);
iSign12.addDirection(Finger.Ring, FingerDirection.DiagonalUpLeft, 1.0);
iSign12.addDirection(Finger.Ring, FingerDirection.DiagonalUpRight, 1.0);

// Pinky
iSign12.addCurl(Finger.Pinky, FingerCurl.NoCurl, 1.0);
iSign12.addDirection(Finger.Pinky, FingerDirection.VerticalUp, 1.0);
```

N_18.js

```
// import dependencies
import {Finger, FingerCurl, FingerDirection, GestureDescription} from 'fingerpose';

// define gesture description
export const nSign18 =  new GestureDescription('н');

// thumb
nSign18.addCurl(Finger.Thumb, FingerCurl.NoCurl, 1.0);
nSign18.addCurl(Finger.Thumb, FingerCurl.HalfCurl, 0.75);
nSign18.addDirection(Finger.Thumb, FingerDirection.DiagonalUpLeft, 1.0);
nSign18.addDirection(Finger.Thumb, FingerDirection.DiagonalUpRight, 1.0);
nSign18.addDirection(Finger.Thumb, FingerDirection.VerticalUp, 0.75);

// index
nSign18.addCurl(Finger.Index, FingerCurl.NoCurl, 1.0);
nSign18.addDirection(Finger.Index, FingerDirection.DiagonalUpLeft, 1.0);
nSign18.addDirection(Finger.Index, FingerDirection.DiagonalUpRight, 1.0);

// middle
nSign18.addCurl(Finger.Middle, FingerCurl.NoCurl, 1.0);
// nSign18.addDirection(Finger.Middle, FingerDirection.VerticalUp, 1.0);Middle
nSign18.addDirection(Finger.Middle, FingerDirection.DiagonalUpLeft, 1.0);
nSign18.addDirection(Finger.Middle, FingerDirection.DiagonalUpRight, 1.0);

// ring
```

```
nSign18.addCurl(Finger.Ring, FingerCurl.FullCurl, 1.0);
nSign18.addCurl(Finger.Ring, FingerCurl.HalfCurl, 0.75);
nSign18.addDirection(Finger.Ring, FingerDirection.VerticalUp, 1.0);

// pinky
nSign18.addCurl(Finger.Pinky, FingerCurl.NoCurl, 1.0);
nSign18.addDirection(Finger.Pinky, FingerDirection.VerticalUp, 1.0);
```

O_19.js

```
// import dependencies
import {Finger, FingerCurl, FingerDirection, GestureDescription} from 'fingerpose';

// define gesture description
export const oSign19 =  new GestureDescription('o');

// thumb
oSign19.addCurl(Finger.Thumb, FingerCurl.NoCurl, 1.0);
oSign19.addCurl(Finger.Thumb, FingerCurl.HalfCurl, 0.75);
oSign19.addDirection(Finger.Thumb, FingerDirection.DiagonalUpLeft, 1.0);
oSign19.addDirection(Finger.Thumb, FingerDirection.DiagonalUpRight, 1.0);

// index
oSign19.addCurl(Finger.Index, FingerCurl.HalfCurl, 1.0);
oSign19.addDirection(Finger.Index, FingerDirection.DiagonalUpLeft, 1.0);
oSign19.addDirection(Finger.Index, FingerDirection.DiagonalUpRight, 1.0);

// middle
oSign19.addCurl(Finger.Middle, FingerCurl.NoCurl, 1.0);
oSign19.addDirection(Finger.Middle, FingerDirection.DiagonalUpLeft, 1.0);
oSign19.addDirection(Finger.Middle, FingerDirection.DiagonalUpRight, 1.0);

// ring
oSign19.addCurl(Finger.Ring, FingerCurl.NoCurl, 1.0);
oSign19.addCurl(Finger.Ring, FingerCurl.HalfCurl, 0.75);
oSign19.addDirection(Finger.Ring, FingerDirection.DiagonalUpLeft, 1.0);
oSign19.addDirection(Finger.Ring, FingerDirection.DiagonalUpRight, 1.0);

// pinky
oSign19.addCurl(Finger.Pinky, FingerCurl.NoCurl, 1.0);
oSign19.addCurl(Finger.Pinky, FingerCurl.HalfCurl, 0.75);
oSign19.addDirection(Finger.Pinky, FingerDirection.DiagonalUpLeft, 1.0);
oSign19.addDirection(Finger.Pinky, FingerDirection.DiagonalUpRight, 1.0);
```

P_20.js

```
// import dependencies
import {Finger, FingerCurl, FingerDirection, GestureDescription} from 'fingerpose';

// define gesture description
export const pSign20 =  new GestureDescription('п');
```

```
// thumb
pSign20.addCurl(Finger.Thumb, FingerCurl.NoCurl, 1.0);
pSign20.addDirection(Finger.Thumb, FingerDirection.VerticalDown, 1.0);

// index
pSign20.addCurl(Finger.Index, FingerCurl.NoCurl, 1.0);
pSign20.addDirection(Finger.Index, FingerDirection.VerticalDown, 1.0);

// middle
pSign20.addCurl(Finger.Middle, FingerCurl.NoCurl, 1.0);
pSign20.addDirection(Finger.Middle, FingerDirection.VerticalDown, 1.0);

// ring
pSign20.addCurl(Finger.Ring, FingerCurl.FullCurl, 1.0)
pSign20.addDirection(Finger.Ring, FingerDirection.VerticalDown, 1.0)

// pinky
pSign20.addCurl(Finger.Pinky, FingerCurl.FullCurl, 1.0)
pSign20.addDirection(Finger.Pinky, FingerDirection.VerticalDown, 1.0)
```

R_21.js

```
// import dependencies
import {Finger, FingerCurl, FingerDirection, GestureDescription} from 'fingerpose';

// define gesture description
export const rSign21 =  new GestureDescription('p');

// thumb
rSign21.addCurl(Finger.Thumb, FingerCurl.HalfCurl, 1.0);
rSign21.addDirection(Finger.Thumb, FingerDirection.DiagonalUpLeft, 1.0);
rSign21.addDirection(Finger.Thumb, FingerDirection.DiagonalUpRight, 1.0);

// index
rSign21.addCurl(Finger.Index, FingerCurl.NoCurl, 1.0);
rSign21.addDirection(Finger.Index, FingerDirection.DiagonalUpLeft, 1.0);
rSign21.addDirection(Finger.Index, FingerDirection.DiagonalUpRight, 1.0);

// middle
rSign21.addCurl(Finger.Middle, FingerCurl.HalfCurl, 1.0);
rSign21.addDirection(Finger.Middle, FingerDirection.DiagonalUpLeft, 1.0);
rSign21.addDirection(Finger.Middle, FingerDirection.DiagonalUpRight, 1.0);

// ring
rSign21.addCurl(Finger.Ring , FingerCurl.NoCurl, 1.0);
rSign21.addDirection(Finger.Ring, FingerDirection.VerticalUp, 1.0);

// pinky
rSign21.addCurl(Finger.Pinky, FingerCurl.NoCurl, 1.0);
rSign21.addDirection(Finger.Pinky, FingerDirection.DiagonalUpLeft, 1.0);
rSign21.addDirection(Finger.Pinky, FingerDirection.DiagonalUpRight, 1.0);
```

S_22.js

```
// import dependencies
import {Finger, FingerCurl, FingerDirection, GestureDescription} from 'fingerpose';
// define gesture description
export const sSign22 =  new GestureDescription('с');

// thumb
sSign22.addCurl(Finger.Thumb, FingerCurl.NoCurl, 1.0);
sSign22.addDirection(Finger.Thumb, FingerDirection.HorizontalLeft, 1.0);
sSign22.addDirection(Finger.Thumb, FingerDirection.HorizontalRight, 1.0);

// index
sSign22.addCurl(Finger.Index, FingerCurl.HalfCurl, 1.0);
sSign22.addDirection(Finger.Index, FingerDirection.HorizontalLeft, 1.0);
sSign22.addDirection(Finger.Index, FingerDirection.HorizontalRight, 1.0);

// middle
sSign22.addCurl(Finger.Middle, FingerCurl.HalfCurl, 1.0);
sSign22.addDirection(Finger.Middle, FingerDirection.HorizontalLeft, 1.0);
sSign22.addDirection(Finger.Middle, FingerDirection.HorizontalRight, 1.0);

// ring
sSign22.addCurl(Finger.Ring, FingerCurl.HalfCurl, 1.0);
sSign22.addDirection(Finger.Ring, FingerDirection.HorizontalLeft, 1.0);
sSign22.addDirection(Finger.Ring, FingerDirection.HorizontalRight, 1.0);

// pinky
sSign22.addCurl(Finger.Pinky, FingerCurl.HalfCurl, 1.0);
sSign22.addDirection(Finger.Pinky, FingerDirection.HorizontalLeft, 1.0);
sSign22.addDirection(Finger.Pinky, FingerDirection.HorizontalRight, 1.0);
```

T_23.js

```
// import dependencies
import {Finger, FingerCurl, FingerDirection, GestureDescription} from 'fingerpose';

// define gesture description
export const tSign23 =  new GestureDescription('п');

// thumb
tSign23.addCurl(Finger.Thumb, FingerCurl.HalfCurl, 0.5);
tSign23.addCurl(Finger.Thumb, FingerCurl.NoCurl, 0.5);
// tSign23.addDirection(Finger.Thumb, FingerDirection.VerticalDown, 1.0);
tSign23.addDirection(Finger.Thumb, FingerDirection.DiagonalDownLeft, 1.0);

// index
tSign23.addCurl(Finger.Index, FingerCurl.NoCurl, 1.0);
tSign23.addDirection(Finger.Index, FingerDirection.VerticalDown, 1.0);

// middle
```

```
tSign23.addCurl(Finger.Middle , FingerCurl.NoCurl, 1.0);
tSign23.addDirection(Finger.Middle, FingerDirection.VerticalDown, 1.0);

// ring

tSign23.addCurl(Finger.Ring , FingerCurl.NoCurl, 1.0);
tSign23.addDirection(Finger.Ring, FingerDirection.VerticalDown, 1.0);

// pinky
tSign23.addCurl(Finger.Pinky, FingerCurl.FullCurl, 1.0);
tSign23.addDirection(Finger.Pinky, FingerDirection.DiagonalDownLeft, 1.0);
tSign23.addDirection(Finger.Pinky, FingerDirection.HorizontalLeft, 0.2);
```

Sh_29.js

```
// import dependencies
import {Finger, FingerCurl, FingerDirection, GestureDescription} from 'fingerpose';

// define gesture description
export const shSign29 =  new GestureDescription('ш');

// thumb
shSign29.addCurl(Finger.Thumb, FingerCurl.HalfCurl, 0.5);
shSign29.addCurl(Finger.Thumb, FingerCurl.NoCurl, 0.5);
shSign29.addDirection(Finger.Thumb, FingerDirection.VerticalUp, 1.0);
shSign29.addDirection(Finger.Thumb, FingerDirection.DiagonalUpLeft, 1.0);

// index, middle, ring
for (let finger of [Finger.Index, Finger.Middle, Finger.Ring]) {
   shSign29.addCurl(finger, FingerCurl.NoCurl, 1.0);
   shSign29.addDirection(finger, FingerDirection.VerticalUp, 1.0);
}

// pinky:
shSign29.addCurl(Finger.Pinky, FingerCurl.FullCurl, 1.0);
shSign29.addDirection(Finger.Pinky, FingerDirection.VerticalDown, 0.2);
shSign29.addDirection(Finger.Pinky, FingerDirection.DiagonalDownLeft, 1.0);
shSign29.addDirection(Finger.Pinky, FingerDirection.HorizontalLeft, 0.2);
```

Soft_33.js

```
// import dependencies
import {Finger, FingerCurl, FingerDirection, GestureDescription} from 'fingerpose';


// define gesture description
export const softSign33 =  new GestureDescription('ь');

// thumb
softSign33.addCurl(Finger.Thumb, FingerCurl.NoCurl, 1.0);
softSign33.addDirection(Finger.Thumb, FingerDirection.HorizontalLeft, 0.25);
```

```
softSign33.addDirection(Finger.Thumb, FingerDirection.HorizontalRight, 0.25);

// index
softSign33.addCurl(Finger.Index, FingerCurl.NoCurl, 1.0);

softSign33.addDirection(Finger.Index, FingerDirection.VerticalUp, 0.25);
// middle, ring, pinky
for (let finger of [Finger.Middle, Finger.Ring, Finger.Pinky]) {
   softSign33.addCurl(finger, FingerCurl.FullCurl, .75);
   softSign33.addDirection(finger, FingerDirection.VerticalUp, 0.25);
}
```

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ імені ІГОРЯ СІКОРСЬКОГО”
Кафедра автоматизованих систем обробки інформації та управління

УЗГОДЖЕНО
Керівник проєкту
_____ *Олена ХАЛУС*
(підпис) (вл. ім’я, прізвище)
“5” квітня2021 р.

ЗАТВЕРДЖУЮ
В.о. завідувача кафедри
_____ *Олександр ПАВЛОВ*
(підпис) (вл. ім’я, прізвище)
“6” квітня2021 р.

СИСТЕМА РОЗПІЗНАВАННЯ ЖЕСТІВ ДЛЯ ВИВЧЕННЯ ДАКТИЛЬНОЇ АБЕТКИ

ТЕХНІЧНЕ ЗАВДАННЯ

Шифр *ДП 7123.01.000 ТЗ*

на 11 сторінках

Київ – 2021 року

ЗМІСТ

1 ЗАГАЛЬНІ ПОЛОЖЕННЯ

1.1 Повне найменування системи та її умовне позначення

Повна назва системи: «Система розпізнавання жестів для вивчення дактильної абетки».

1.2 Найменування організації-замовника та організації-учасника робіт

Замовником є кафедра автоматизованих систем обробки інформації та управління факультету інформатики та обчислювальної техніки Національного технічного університету України "Київський політехнічний інститут ім. Ігоря Сікорського" (далі за текстом — Замовник).

Адреса замовника: м. Київ, п. Перемоги 37, 18 корпус ФІОТ.

Розробник веб-додатку — Руденко Анастасія Олександрівна (далі за текстом – Виконавець), студентка групи ІС-71 Кафедри автоматизованих систем обробки інформації та управління Факультету інформатики та обчислювальної техніки Національного технічного університету України "Київський політехнічний інститут ім. Ігоря Сікорського".

1.3 Перелік документів, на підставі яких створюється система

Розробка системи і створення проектно-експлуатаційної документації проводиться Виконавцем відповідно до стандартів:

- ДСТУ 34.601-90. Комплекс стандартів на автоматизовані системи. Автоматизовані системи. Стадії створення;
- ДСТУ 19.201-78. Технічне завдання. Вимоги до змісту і оформлення;
- ДСТУ 34.201-89. Інформаційні технології. Комплекс стандартів на автоматизовані системи. Види, комплексність і позначення документів при створенні автоматизованих систем.

1.4 Планові терміни початку і закінчення роботи зі створення системи

Плановий термін початку роботи над розробкою застосунку– 12 квітня 2021 року. Плановий термін по закінченню роботи над розробкою застосунку – 14 травня 2021 року.

2 ПРИЗНАЧЕННЯ І ЦІЛІ СТВОРЕННЯ СИСТЕМИ

2.1 Призначення системи

Система розпізнавання жестів для вивчення дактильної абетки призначена для поліпшення умов соціалізації людей з вадами слухового чи/або мовного апарату.

Зараз існує безліч додатків, які дозволяють користувачам вивчати іноземні мови. Вони дозволяють практикувати вимову та отримувати негайні відгуки від програми. Але така функціональність майже не представлена для вивчення дактильної абетки або мови жестів.

Систему розпізнавання жестів для вивчення дактильної абетки відкриває можливість інтерактивного процесу навчання, що є важливим етапом на шляху до підвищення якості комунікації між різними людьми.

2.2 Цілі створення системи

Для досягнення мети розробки системи розпізнавання жестів для вивчення дактильної абетки наступні задачі потрібні бути реалізованими:

а) можливість відтворення відео-потоку з камери комп’ютера;

б) можливість формування завдання для користувача:

1) вибір завдання (жесту) системою:

– отримання комплексного завдання на всі літери;

– отримання завдання у вигляді однієї рандомної літери;

2) вибір завдання (жесту) користувачем:

– введення літери-завдання з клавіатури;

– вибір літери-завдання зі списку;

в) відображення учбового матеріалу у відповідності до завдання;

г) розпізнавання однієї людської долоні з відео-потоку;

г) відображення 21 ключової точки руки у реальному часі на відео;

д) розпізнавання жесту, відтвореного користувачем за допомогою однієї долоні;

е) надання оцінки про виконання користувачем потрібного жесту.

3 ХАРАКТЕРИСТИКА ОБ'ЄКТА АВТОМАТИЗАЦІЇ

Об'єктом автоматизації є процес вивчення дактильної абетки.

Вхідні дані – відео-потік, отриманий за допомогою камери пристрою користувача. Користувач, знаходячись перед веб-камерою, відтворює жест-завдання.

Результатом роботи системи є оцінка відтворення жесту користувачем (відтворено/не відтворено), отримана після співставлення жесту користувача з жестом-завданням.

4 ВИМОГИ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

4.1 Вимоги до функціональних характеристик

Система розпізнавання жестів для вивчення дактильної абетки повинна відповідати наступним функціональним вимогам:

а) відтворення відео-потоку з камери комп’ютеру;

б) надання системою завдання (жесту) для відтворення користувачем;

в) вибір користувачем завдання (жесту) для відтворення користувачем;

г) відображення відповідного учбового матеріалу (зображення жесту) до завдання;

ґ) розпізнавання однієї людської долоні з відео-потоку;

д) відображення 21 ключової точки руки у реальному часі;

е) розпізнавання жесту, який показано за допомогою однієї долоні;

є) надання оцінки про виконання користувачем потрібного жесту.

4.2 Вимоги до надійності

Система повинна оброблювати всі виключні ситуації, що пов’язані знекоректними отриманими даними.

4.3 Вимоги до видів забезпечення

Для коректної роботи системи розпізнавання жестів для вивчення дактильної абетки повинні виконуватися наступні вимоги до апаратного забезпечення:

а) процесор з тактовою частотою від 2.3 ГГц;

б) оперативна пам’ять від 8 ГБ.

Обов’язкова комп’ютерна периферія - камера.

Додаткові умови роботи системи розпізнавання жестів для вивчення дактильної абетки:

а) операційна система MacOS 11.0, Windows 10;

б) підключення до мережі інтеренет;

в) встановлений браузер Safari / Google Chrome / Mozilla Firefox.

5 СТАДІЇ І ЕТАПИ РОЗРОБКИ

Таблиця 5.1 – Етапи розробки

№ п/п	Назва етапу	Термін виконання
1.	Вивчення рекомендованої літератури	09.04.2021
2.	Проведення ґрунтовного аналізу аналогів	12.04.2021
3.	Формування технічного завдання	20.04.2021
4.	Узгодження сценарію розробки з керівником	23.04.2021
5.	Проєктування архітектури веб-додатку	28.04.2021
6.	Розробка алгоритму розпізнавання руки	30.04.2021
7.	Формування жестів дактильної абетки	05.05.2021
8.	Розробка алгоритму розпізнавання жесту	08.05.2021
9.	Розробка інтерфейсу для навчання	10.05.2021
10.	Налагодження програмного продукту	12.05.2021
11.	Оформлення ПЗ	14.05.2021

6 ПОРЯДОК КОНТРОЛЮ ТА ПРИЙМАННЯ

6.1 Види випробувань

Для того щоб упевнитися в коректному функціонуванні розробленого веб-застосування проводиться функціональне тестування. Цей тип випробувань визначає відповідність програмного продукту встановленим функціональним вимогам.

UNICHECK

Ім'я користувача:
Попенко Володимир Дмитрович

Дата перевірки:
03.06.2021 14:43:46 EEST

Дата звіту:
03.06.2021 17:33:36 EEST

ID перевірки:
1008158042

Тип перевірки:
Doc vs Internet + Library

ID користувача:
77149

Назва документа: **Rudenko_bachelor_is71**

Кількість сторінок: **46** Кількість слів: **4529** Кількість символів: **34451** Розмір файлу: **6.88 MB** ID файлу: **1008224497**

2.78%
Схожість

Найбільша схожість: **1.39%** з джерелом з Бібліотеки (ID файлу: **1000037703**)

1.41% Джерела з Інтернету	7	Сторінка 48
2.78% Джерела з Бібліотеки	109	Сторінка 48

0% Цитат

Не знайдено жодних цитат

Вилучення списку бібліографічних посилань вимкнене

0%
Вилучень

Немає вилучених джерел

Модифікації

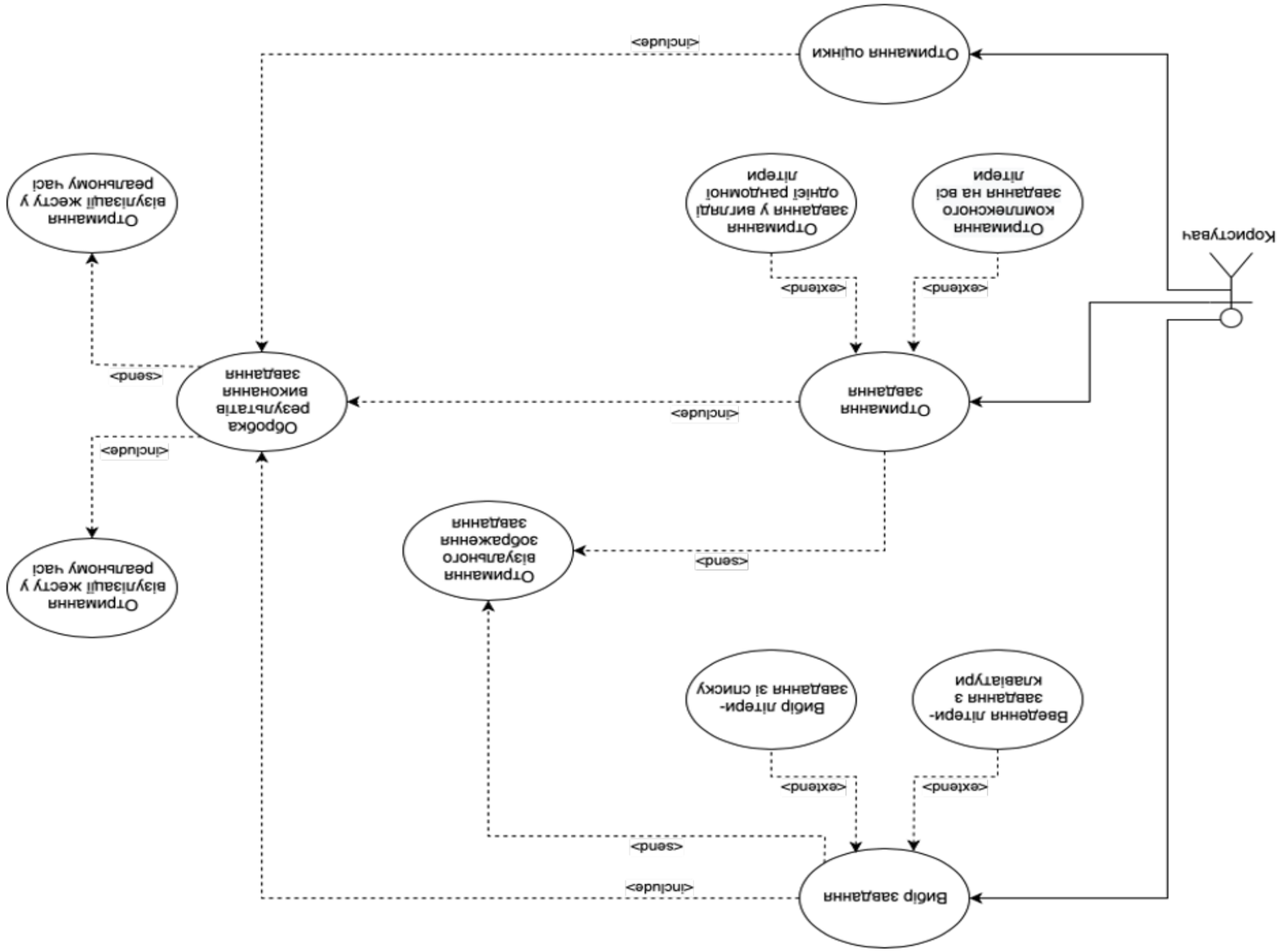
Виявлено модифікації тексту. Детальна інформація доступна в онлайн-звіті.

Замінені символи	4

Графічний матеріал
до дипломного проєкту

на тему: «Система розпізнавання жестів для вивчення дактильної абетки»

Київ – 2021 року



Зм.	Арк.	№ документа	Підпис	Дата
Розробила		Руденко А.О.		
Перевірила		Харус О.А.		
Консульт.				
Н/контро		Жураковська О.С.		
Затвердила		Харус О.А.		

ДП 7123.02.000 ССВ

Літера	Маса	Масштаб

Схема структурна

варіантів використань

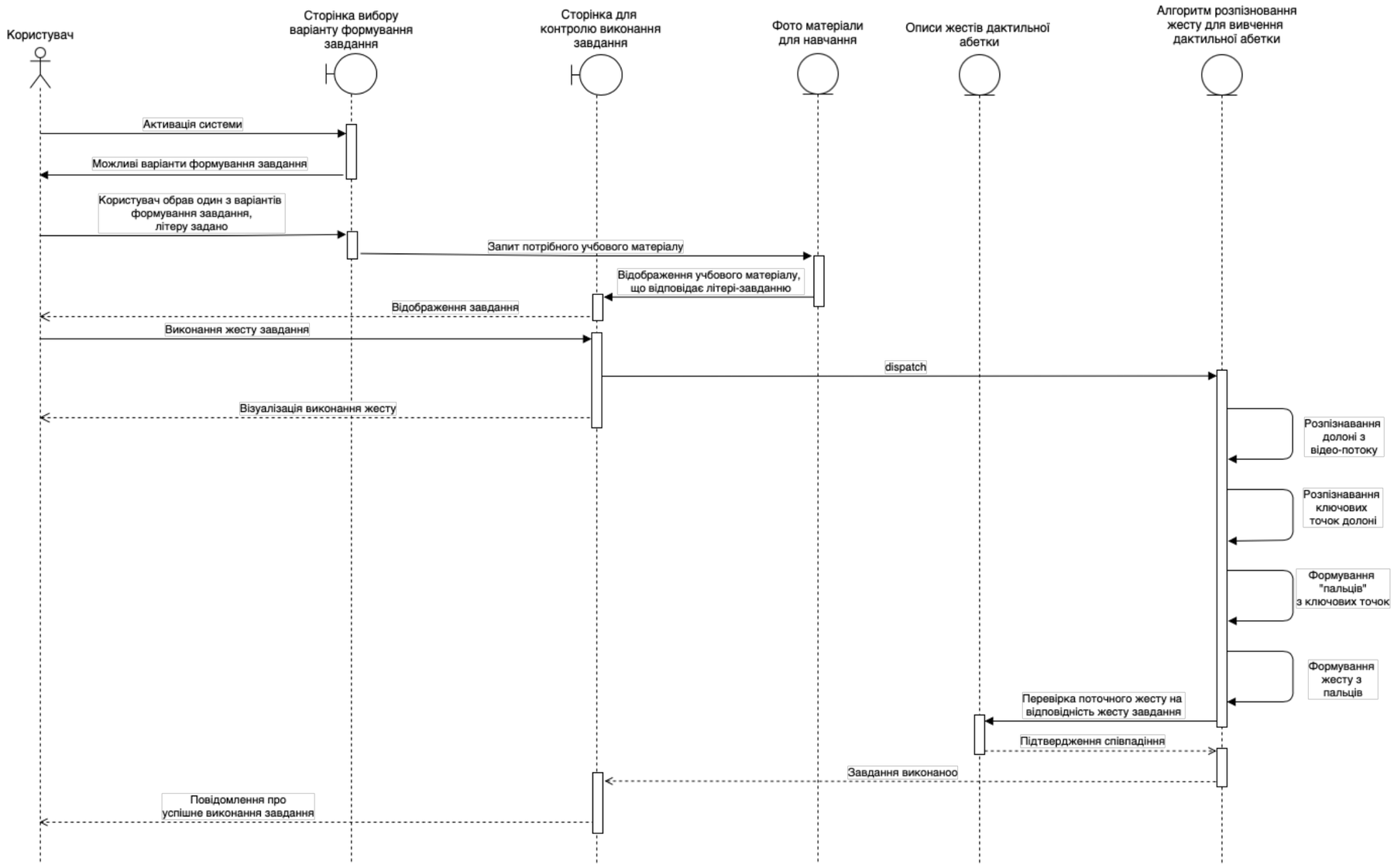
Аркуш 1	Аркушів 1

КПІ ім. Ігоря Сікорського
Каф. АСОІУ
Гр. ІС-71

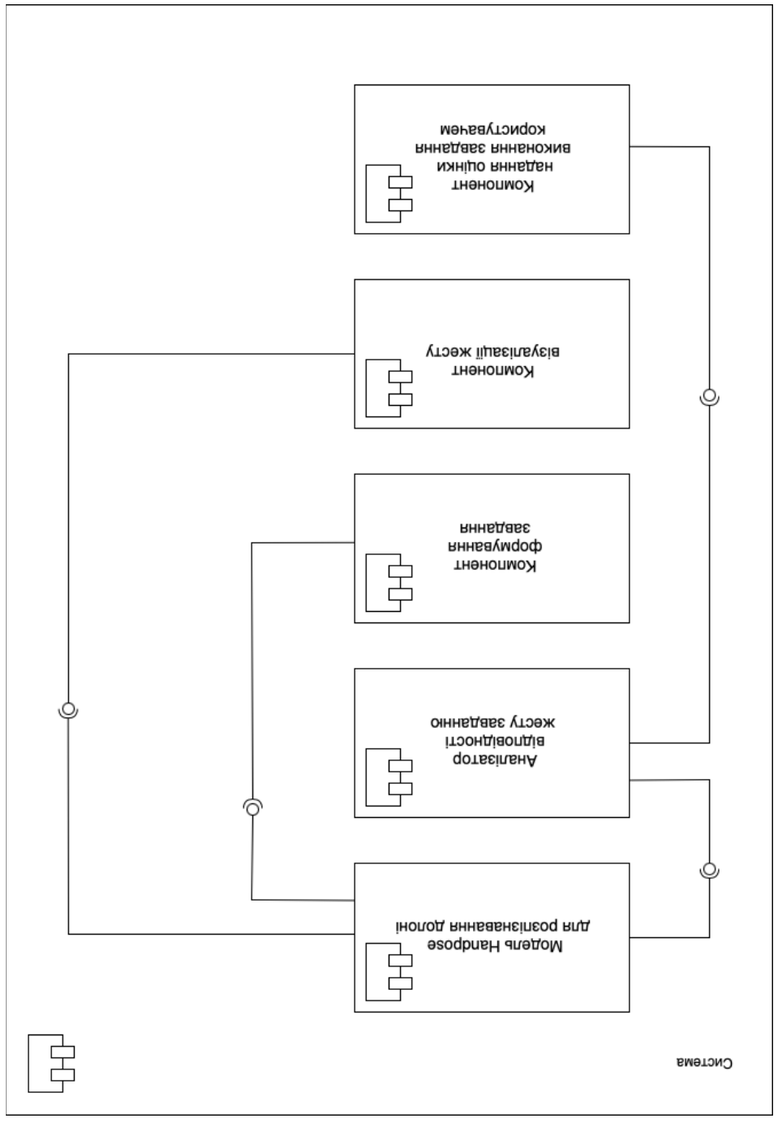
Система розпізнавання жестів для вчення дактильної абетки

ДП 7123.02.000 ССВ

ДП 7123.03.000 ССП



					ДП 7123.03.000 ССП		
					Схема структурна послідовності	Літера	Маса / Масштаб
Зм.	Арк.	№ документа	Підпис	Дата			
Розробила		Руденко А.О.				Аркуш 1	Аркушів 1
Перевірила		Халус О.А.					
Консульт.					Система розпізнавання жестів для вивчення дактильної абетки	КПІ ім. Ігоря Сікорського Каф. АСОІУ Гр. ІС-71	
Н/контро		Жураковська О.С.					
Затвердила		Халус О.А.					



Зм.	Арк.	№ документа	Підпис	Дата						
Розробила		Руденко А.О.			*Схема структурна компонентів*			Літера	Маса	Масштаб
Перевірила		Харус О.А.								
Консульт.								Аркуш 1		Аркушів 1
Н/контро		Журавковська О.С.								
Затвердила		Харус О.А.			*Система розпізнавання жестів для вдосконалення дактильної абетки*			*КПІ ім. Ігоря Сікорського Каф. АСОІУ Гр. ІС-71*		

ДП 7123.04.000 СК

ДП 7123.04.000 СК

Вітаємо у системі розпізнавання жестів для вивчення дактильної абетки!

Оберіть зручний для Вас спосіб отримання завдання

Самостійно | Від системи

Спосіб введення завдання

Обрати зі списку | Ввести самостійно

Оберіть літеру-завдання

а | До виконання!

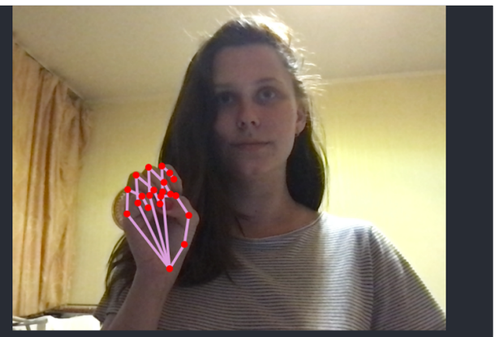
Введіть літеру-завдання

а

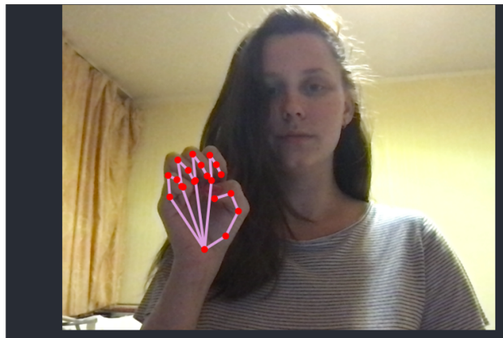
До виконання!

Тип завдання

Одна рандомна літера | Усі літери поспіль



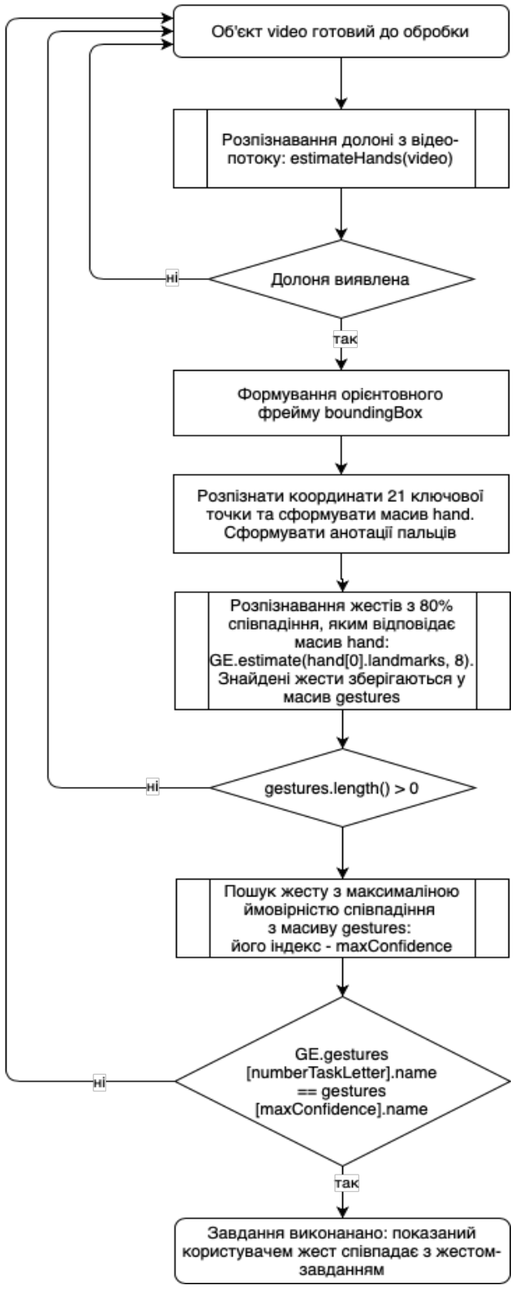
Покажіть літеру, як на картинці!



Вірно! Ви показали а

					ДП 7123.05.000 КЕ
Зм.	Арк.	№ документа	Підпис	Дата	Креслення вигляду екранних форм
Розробила		Руденко А.О.			Літера: — Маса: — Масштаб: —
Перевірила		Халус О.А.			Аркуш 1 / Аркушів 1
Консульт.					Система розпізнавання жестів для вивчення дактильної абетки
Н/контро		Жураковська О.С.			КПІ ім. Ігоря Сікорського, Каф. АСОІУ, Гр. ІС-71
Затвердила		Халус О.А.			

Рішення з математичного забезпечення



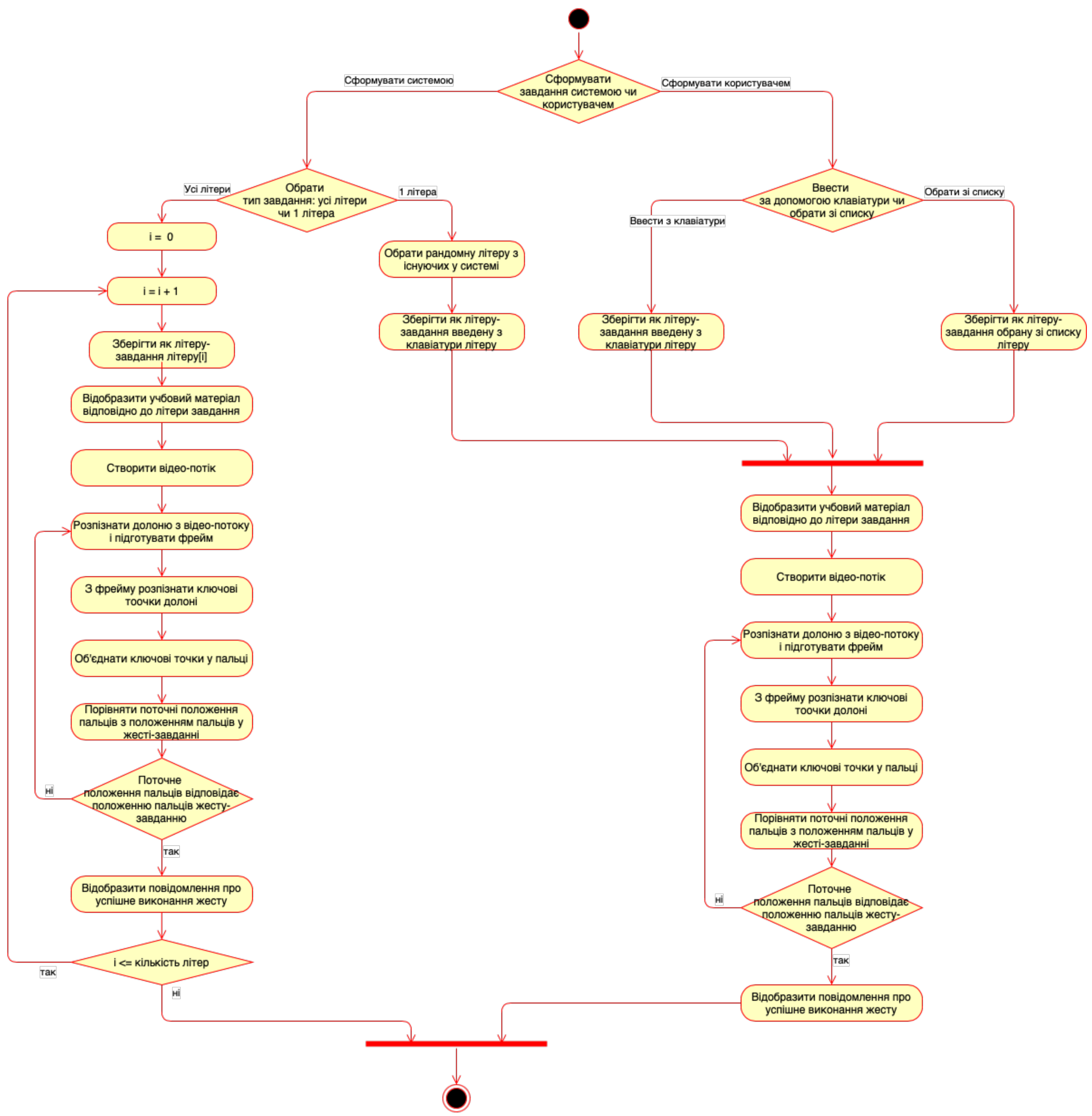
Постановка задачи

Користувач системи на камеру свого комп'ютеру показує жест-завдання. Для підтвердження того, що жест, відтворений користувачем дійсно є жестом-завданням, система використовує зображений алгоритм розпізнавання жесту для формування оцінки виконання жесту-завдання

Демонстраційний плакат до дипломного проекту
«Система розпізнавання жестів для вивчення дактильної абетки»

Виконала студентка гр. ІС-71	Руденко А.О.
Керівник ДП	Халус О.А.

ДП 7123.06.000 ССД



					ДП 7123.06.000 ССД
Зм.	Арк.	№ документа	Підпис	Дата	
Розробила		Руденко А.О.			Схема структурна діяльності
Перевірила		Халус О.А.			
Консульт.					Система розпізнавання жестів для вивчення дактильної абетки
Н/контро		Жураковська О.С.			
Затвердила		Халус О.А.			

Літера	Маса	Масштаб
Аркуш 1	Аркушів 1	

КПІ ім. Ігоря Сікорського
Каф. АСОІУ
Гр. ІС-71