

SAPIENTIA ERDÉLYI MAGYAR TUDOMÁNYEGYETEM

Marosvásárhelyi Kar



TDK dolgozat

Gesture Vision AI: gesztusalapú interakciók és események valós idejű menedzselése

Témavezető

Dr. ICLĂNZAN David

Docens

Hallgató

TÓFALVI Patrik

Informatika

2025

Kivonat

A dolgozat egy valós idejű gesztusfelismerő rendszer fejlesztését mutatja be, amely kézmozdulatok és egyéb emberi gesztusok detektálására és azok eseményekké alakítására szolgál. A rendszer célja, hogy rugalmas és könnyen integrálható megoldást nyújtson különböző alkalmazási területeken, például interaktív prezentációk vezérlésére, okosotthon-rendszerek irányítására, vagy értesítések küldésére mobilkészülékekre.

A fejlesztett API valós időben képes azonosítani a felhasználó gesztusait és azokat webhookokon keresztül továbbítani a kliensoldali alkalmazások számára. A kapott események szabadon feldolgozhatók, lehetőséget biztosítva különféle interakciós modellek kialakítására. Az API webhook-alapú architektúrája lehetővé teszi a széleskörű integrációt különböző platformokon, beleértve webes alkalmazásokat, mobilrendszereket és IoT-eszközöket. A rendszer nyitott kialakítása lehetőséget biztosít a fejlesztők számára további gesztusok hozzáadására és testreszabott interakciós modellek kidolgozására.

Dolgozatomban a rendszer egyik kiemelt alkalmazási területe az interaktív oktatás és prezentációk, ahol az előadó kézmozdulatai révén navigálhat a vetített tartalomban, sőt akár vizuálisan is integrálható a prezentációba, így fokozva az élményt és a figyelem fenntartását.

Az eredmények igazolják a rendszer hatékonyságát és rugalmasságát. Az API egyszerűsíti a természetes ember-gép interakciók fejlesztését és integrálását számos gyakorlati alkalmazásban.

Kulcsszavak: számítógépes látás, gesztusdetektálás, eseményalapú interakciók

Tartalomjegyzék

Ábrák jegyzéke	IV
1. Bevezető	1
2. Szakirodalmi áttekintés	4
2.1. Számítógépes látás (Computer Vision)	4
2.2. Mesterséges intelligencia (AI) és gépi tanulás	6
2.3. A MediaPipe működése és kézcsomópontok detektálása	8
2.4. Hasonló alkalmazások és fejlesztések	9
3. Tervezés	12
3.1. Célkitűzések	12
3.2. Felhasználói igények	13
3.3. Rendszer követelményei	13
3.4. Tervezési alapelvek és architekturális megfontolások	14
4. Rendszer bemutatása	16
4.1. Gesztusfelismerő API	17
4.2. Kliens alkalmazások	19
4.2.1. Oktatást segítő prezentációs alkalmazás	19
4.2.2. Gesztusvezérelt okos lámpa	23
4.2.3. Push értesítések mobil eszközre	23
5. Megvalósítás	25
5.1. Gesztusfelismerő API	25
5.1.1. Webhook kezelés	25
5.1.2. Képkockák feldolgozása	27
5.1.3. Gesztus feldolgozása és továbbítása	27
5.1.4. Gesztusfelismerés a MediaPipe segítségével	27
5.1.5. Webhook-alapú értesítések	30
5.2. Oktatást segítő prezentációs alkalmazás	31

5.2.1.	Feliratkozás az API gesztusfelismerésére	31
5.2.2.	Értesítés fogadása az API-tól	32
5.2.3.	Kép szegmentálása	33
5.2.4.	Prezentációs fájlok kezelése	35
5.2.5.	Szegmentált kép megjelenítése a vásznon	36
5.2.6.	Gesztus alapú interakciók	39
5.2.7.	Laser pointer	39
5.2.8.	i18n - Többnyelvűség támogatása	41
5.2.9.	Funkciók beállítása a felismert gesztusra	42
5.2.10.	Gyakorló felület	43
5.3.	Gesztusvezérelt okos lámpa	44
5.3.1.	Használt hardverelemek és bekötések	44
5.3.2.	Bekötési logika és biztonsági megfontolások	45
5.3.3.	ESP8266, NodeMCU és GPIO lábak szerepe	45
5.3.4.	A Wi-Fi alapú kommunikáció előnyei	46
5.3.5.	Miért ezeket az eszközöket használtam?	47
5.3.6.	Kódrészlet	47
5.4.	Push értesítés küldése mobiltelefonra	48
5.4.1.	Feliratkozás az API-ra	48
5.4.2.	Értesítés fogadása az API-tól	49
5.4.3.	Értesítés megjelenítése	50
6.	Mérések	52
6.1.	Latencia(kliens - API között)	52
6.2.	Gesztusfelismerés pontossága	55
6.3.	Fényviszonyok	58
6.4.	Erőforrások	61
7.	Eredmények tárgyalása	63
7.1.	Latencia elemzése	63
7.2.	Gesztusfelismerés pontossága	63
7.3.	Fényviszonyokra való érzékenység	64
7.4.	Erőforrásigény és teljesítmény	64
8.	Következtetések	65
8.1.	Továbbfejlesztési lehetőségek	66
Irodalom		67

Ábrák jegyzéke

2.1. MediaPipe kézcsomópontok (landmarks)	5
2.2. Konvolúciós neurális hálózat (CNN)	7
2.3. MediaPipe Palm Detection működése	9
4.1. A rendszer működése	16
4.2. Az API folyamatábrája	17
4.3. Kézcsomópontok megjelenítése (debug mód)	18
4.4. Szekvencia diagram	20
4.5. Use Case diagram	21
4.6. API - kliens kapcsolat	22
4.7. Gesztusvezérelt lámpa architektúrája	23
4.8. A feliratkozott kliensek értesítést kapnak gesztus felismerés után . .	24
5.1. Az alkalmazás Presentation oldala	38
5.2. Laser pointing, ujj használata mutatóra, egér kurzor nélkül	41
5.3. Az alkalmazás Beállítások oldal	42
5.4. Az okos lámpa kapcsolási rajza	46
5.5. Értesítés iOS mobiltelefonon	51
6.1. Latencia, átlag és szórás	53
6.2. Boxplot diagram	54
6.3. Hisztogram	54
6.4. A log fájl tartalma statikus gesztus esetén	55
6.5. A log fájl tartalma dinamikus gesztus esetén	56
6.6. Konfúziós mátrix a „Swipe Right” gesztus felismeréséről	57
6.7. Chi-négyzet próba a felismerések összehasonlítására	58
6.8. A telefonos alkalmazással mért lux mennyiség, szobában természetes fény és lámpa megvilágítással	59
6.9. Természetes fényben mért megvilágítás	59
6.10. Szobában mért megvilágítás	60
6.11. Gyenge fényviszonyok	60

6.12. Sötétben a szegmentálás romlása	61
---	----

1. fejezet

Bevezető

Az elmúlt évtizedekben jelentős fejlődésen ment keresztül a természetes ember-gép interakció, köszönhetően a számítógépes látás, mesterséges intelligencia és a szenzorok rohamos fejlődésének [1]. A felhasználói élmény javítása érdekében egyre nagyobb igény mutatkozik olyan megoldásokra, amelyek lehetővé teszik az érintésmentes vezérlést és a természetesebb interakciót a digitális eszközökkel. A gesztusvezérlés egy ilyen megoldás, amely lehetőséget biztosít arra, hogy a felhasználók kézmozdulataikkal, ujjmozdulataikkal vagy akár testtartásukkal vezéreljenek különböző rendszereket.

A gesztusfelismerés egyik legfontosabb előnye, hogy lehetővé teszi az érintésmentes interakciót, ami különösen hasznos olyan környezetekben, ahol a fizikai érintés nem praktikus vagy nem megengedett. Ilyen területek lehetnek például az egészségügy (pl. műtői rendszerek vezérlése steril környezetben), az okosotthonok (pl. világítás vagy háztartási eszközök irányítása), az autópár (pl. multimédiás rendszerek kezelése vezetés közben), valamint az oktatás és a prezentációk területén, ahol a gesztusvezérlés fokozhatja az előadások interaktivitását [2, 3].

A dolgozat kutatási fókusza annak vizsgálata, hogyan valósítható meg egy rugalmas, valós idejű gesztusfelismerő API (Application Programming Interface), amely webhook-alapú eseményvezérelt architektúrát alkalmaz, és hatékonyan integrálható különböző platformokra, különös tekintettel az interaktív oktatási és prezentációs környezetekre. A fejlesztés újdonsága a hagyományos, folyamatos polling-alapú megoldásokkal szemben az, hogy az API kizárólag események bekövetkezésekor kommunikál a kliensalkalmazásokkal. Ez az architektúra jelentősen csökkenti a kliensoldali számítási és hálózati terhelést, miközben lehetővé teszi a rendszer valós idejű válaszidejét és platformfüggetlen integrálhatóságát. A kutatás célja annak feltérképezése, hogy ez az architektúra milyen előnyöket kínál a jelenleg elérhető gesztusfelismerő rendszerekhez képest, és milyen módon segítheti elő a természetesebb ember-gép interakciót (HCI – Human-Computer Interaction) az oktatásban és más alkalmazási területeken.

A gesztusfelismerő API működése során a kliensalkalmazás folyamatosan képkockákat (frame-eket) küld az API egyik végpontjára (endpointjára). Az API ezeket a képeket valós időben feldolgozza és mesterséges intelligencia alapú műveleteket végez rajtuk, vagyis kézdetektálást. A felismerési folyamat során az API azonosítja a felhasználó által végzett gesztusokat, majd ha a kliens előzetesen feliratkozott egy webhookra, az API automatikusan értesítést küld számára a detektált gesztusról. A kliensoldali alkalmazás ezután tetszőleges módon felhasználhatja az értesítést, például egy prezentáció irányítására, egy okosotthon eszköz vezérlésére vagy egy figyelmeztetés küldésére egy mobil eszközre.

A fejlesztett rendszer egyik kiemelt alkalmazási területe az interaktív oktatás és prezentációk. Az online és hibrid oktatás elterjedtével egyre nagyobb szükség van olyan megoldásokra, amelyek segítik az előadók munkáját, miközben fokozzák a hallgatóság figyelmét és vizuális élményét. A gesztusvezérlés lehetőséget biztosít arra, hogy az előadó kézmozdulatokkal navigáljon a prezentációban, lapozzon a diák között, vagy akár saját magát is vizuálisan integrálja a bemutatóba.

A COVID-19 pandémia idején, amikor az oktatás nagyrészt online térbe helyeződött át, az ilyen interaktív megoldások kiemelt szerepet kaptak. Az oktatók számára nagy kihívást jelentett a hallgatóság figyelmének fenntartása és az előadások hatékony lebonyolítása a digitális platformokon keresztül [4]. A gesztusvezérlés ebben az időszakban nagy előnyt jelentett, hiszen lehetővé tette az érintésmentes és természetes interakciót a prezentációs anyagokkal, így az előadók dinamikusabban és szemléletesebben tudták közvetíteni az információkat. A hallgatók számára is előnyös volt, hogy az oktatók gesztusokkal tudták irányítani az órát, ami vizuálisan érdekesebbé és érthetőbbé tette az online tananyagokat [5].

A rendszer rugalmasságát növeli, hogy a felhasználók testre szabhatják a gesztusvezérlést, vagyis beállíthatják, hogy egy-egy gesztus milyen műveletet hajtson végre az alkalmazáson belül. Ezen felül a rendszer tartalmaz egy virtuális lézermutatót is, amelyet a felhasználó mutatóujjával irányíthat. Ez lehetőséget biztosít arra, hogy az előadó a prezentáció egyes elemeire rámutasson, kiemeljen fontos részeket, vagy akár kurzorként használja mutogatásra. A rendszer hatékonyabb használata érdekében egy külön gyakorlófelület is rendelkezésre áll, ahol a felhasználók előzetesen kipróbálhatják és begyakorolhatják a gesztusokat, így a prezentáció közben gördülékenyen és magabiztosan alkalmazhatják azokat. A széleskörű felhasználhatóság érdekében a kliensalkalmazás három nyelven érhető el: magyarul, románul és angolul, így biztosítva a különböző nyelvi háttérű felhasználók számára az akadálymentes használatot.

A rendszer azonban nem kizárólag az oktatásban használható, hanem számos más környezetben is alkalmazható. Az okosotthonok esetében például lehetőséget biztosít a világítás, a fűtés vagy az audio rendszerek érintésmentes vezérlésére [6]. A fejlesztés során egy konkrét alkalmazási példára is kitérnek: egy gesztusvezérléssel működő okoslámpára. A rendszer egy mikrovezérlővel ellátott asztali lámpát irányít,

amely képes internetkapcsolatra, és az API webhookjára feliratkozva fogadja a gesztusfelismerési értesítéseket. A detektált gesztusok alapján a lámpa be- vagy kikapcsolható, demonstrálva ezzel a gesztusvezérlés gyakorlati hasznosíthatóságát.

Ezen kívül ipari környezetben akár robotkarok vagy más automata rendszerek irányítására is alkalmas lehet [7]. Emellett a rendszer egyszerű webhook-alapú értesítései lehetővé teszik, hogy a gesztusokat értesítések küldésére is felhasználjuk, például egy bizonyos mozdulat végrehajtása után a felhasználó értesítést kapjon a mobiltelefonján.

2. fejezet

Szakirodalmi áttekintés

A gesztusvezérlés és a számítógépes látás területe az interaktív ember-gép interakció (HCI – Human-Computer Interaction) kutatásának fontos része. A számítógépes látás, mesterséges intelligencia és a gépi tanulás alapelvei mind kulcsfontosságúak a gesztusok felismerésében. A modern informatikai rendszerek egyik fő célja, hogy a felhasználók számára minél természetesebb, intuitívabb módon biztosítsanak interakciós lehetőségeket a digitális eszközökkel. A gesztusvezérlés lehetőséget nyújt arra, hogy a fizikai beviteli eszközöket (például egér, billentyűzet) érintésmentes, kézmozdulat-alapú irányítás váltsa fel, ezáltal növelve a felhasználói élményt és a hatékonyságot különböző szituációkban. A tudományos háttérhez az alábbi témák és kutatások tartoznak:

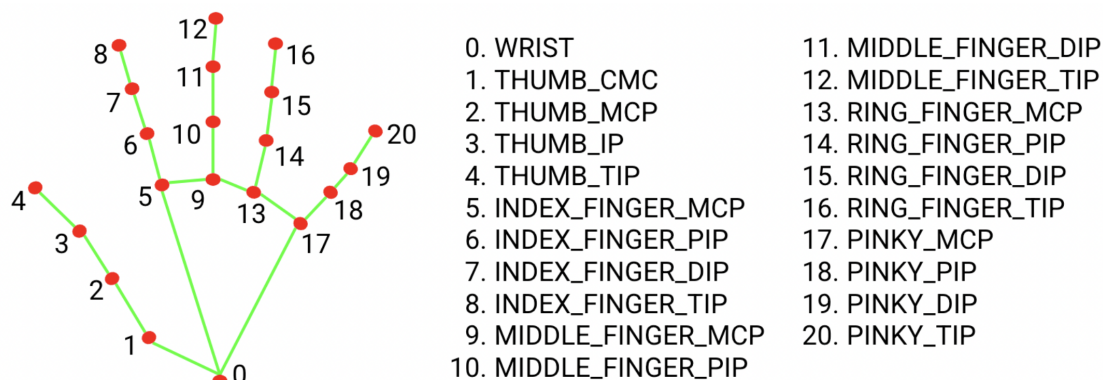
2.1. Számítógépes látás (Computer Vision)

A számítógépes látás az a technológia, amely lehetővé teszi, hogy a gépek a vizuális világot észleljék és értelmezzék. A gesztusvezérlés esetében a képfeldolgozás és a tárgy- és mozgásfelismerés elengedhetetlen [8]. A képfeldolgozás során általában az alábbi lépések történnek:

- Kép előfeldolgozás: zajszűrés, kontrasztjavítás
- Detektálás: például kéz vagy arc lokalizálása (bounding box)
- Kivonatolás: jellemző pontok (pl. ujjhegyek, ízületek) azonosítása
- Következtetés: gesztus vagy mozgás osztályozása

A kéz- és testmozdulatok valós idejű követésére ma már széles körben alkalmazzák a Google által fejlesztett MediaPipe keretrendszert, amely képes 21 kézcsomópontot (landmark) azonosítani (lásd 2.1. ábra) egyetlen képkockán belül [9]. A MediaPipe

az egyik legelterjedtebb nyílt forráskódú eszköz a valós idejű gesztusfelismerés terén, és optimalizált mobil- és webes alkalmazásokhoz is.



2.1. ábra. MediaPipe kézcsomópontok (landmarks)

Forrás: www.ai.google.dev

A MediaPipe mellett számos egyéb számítógépes látás alapú megoldás is létezik a gesztus- és mozdulatfelismerésre, amelyek különböző előnyöket kínálnak:

- **OpenPose** – A Carnegie Mellon University által fejlesztett, nyílt forráskódú rendszer, amely képes több személy test-, kéz- és arcpontjainak egyidejű detektálására. Különösen alkalmas komplex feldolgozására, azonban nagy számítási igénye miatt főként erőforrásigényes környezetekben használják [10].
- **YOLO (You Only Look Once)** – Egy gyors objektumdetektáló modell, amely kéz- vagy testpozíciókat bounding box-al képes felismerni, és kategorizálni a gesztusokat (pl. „mutatás”, „tenyér nyitva”). Gyorsasága miatt valós idejű alkalmazásokban népszerű, különösen beágyazott rendszereken [11].
- **Saját tanított modellek (CNN + LSTM)** – Dinamikus gesztusok felismeréséhez gyakran használnak kombinált neurális hálózatokat, ahol a CNN-ek(Convolutional Neural Networks) a képekből kivonatolt jellemzőket, az LSTM-ek(Long Short-Term Memory) pedig ezek időbeli változását elemzik. Ezek a modellek jól alkalmazhatók időfüggő mozdulatok felismerésére [12].
- **3D szenzoros megoldások (pl. Microsoft Kinect, Intel RealSense)** – Ezek a rendszerek mélységkamerát használnak, és 3D koordinátákat szolgáltatnak a kéz, test vagy fej mozgásairól. Ipari és egészségügyi alkalmazásokban gyakran használatosak, mivel pontos és részletes mozgásrekonstrukcióra képesek [13].
- **Vision Transformers (ViT)** – A legújabb kutatásokban megjelentek a

transformer alapú modellek, amelyek képesek a képek globális és lokális összefüggéseit hatékonyan megtanulni. Bár ezek a modellek még nem terjedtek el széles körben a valós idejű rendszerekben, kutatási célokra és nagy pontosságot igénylő feladatokra egyre gyakrabban használják [14].

Ezek a megoldások különböző technikai előnyöket kínálnak, a választás pedig leginkább az adott alkalmazás környezetétől és teljesítményigényétől függ. A dolgozatomban alkalmazott MediaPipe keretrendszer előnye, hogy gyors, erőforrás-hatékony és jól integrálható webes vagy mobil környezetekbe, így ideális választás valós idejű gesztusvezérléshez.

2.2. Mesterséges intelligencia (AI) és gépi tanulás

A mesterséges intelligencia az informatika egyik legjobban fejlődő területe, amely lehetővé teszi, hogy a számítógépek emberi intelligenciához hasonló döntéshozatali folyamatokat hajtsanak végre. Ennek egyik legfontosabb ága a gépi tanulás (Machine Learning, ML), amely során az algoritmusok képesek tanulni a korábban látott adatokból, és ezek alapján új helyzetekben is döntéseket hozni.

A gépi tanulás különböző módokon valósulhat meg:

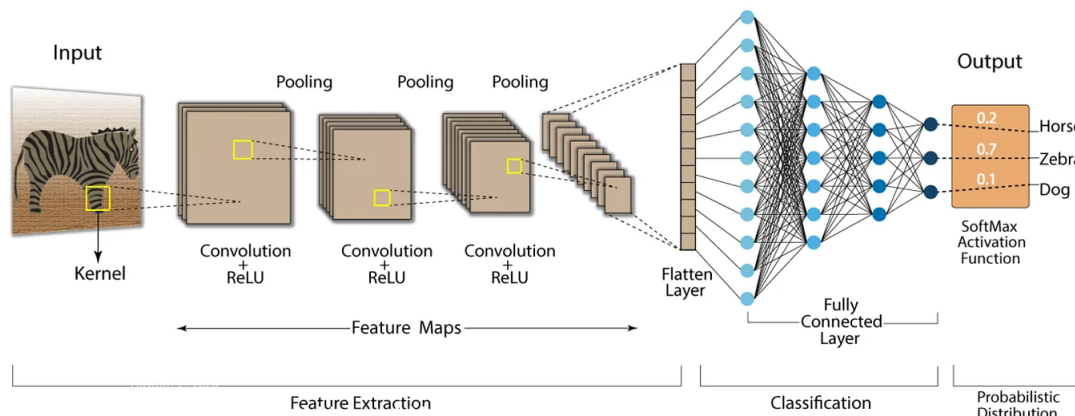
- **Felügyelt tanulás** - Az algoritmus bemenetként adathalmazt és az ahhoz tartozó címkéket (pl. kézmozdulat típusa) kap, és ezek alapján megtanulja az adatok és címkék közötti összefüggéseket.
- **Nem felügyelt tanulás** - Az adatokhoz nem tartoznak címkék, az algoritmusnak magának kell felismernie a mintázatokat vagy klasztereket.
- **Megerősítő tanulás** - Az algoritmus egy környezettel lép interakcióba, és jutalmak vagy büntetések alapján tanulja meg az optimális döntési stratégiát.

A gesztusfelismerés területén leggyakrabban a felügyelt tanulást alkalmazzák, hiszen szükség van arra, hogy a kézmozdulatokhoz konkrét osztályok (pl. „mutat”, „tenyér nyitva”, „összecsíp”) legyenek rendelve.

A mélytanulás (Deep Learning) a gépi tanulás egyik ága, amely többrétegű neurális hálózatokat alkalmaz. Ezek a hálózatok képesek hierarchikus jellemzők (features) automatikus kivonására az adatokból. A gesztusfelismerés során például a kéz kontúrja, ujjhossz, ízületi elhelyezkedések – mind automatikusan detektálhatók a hálózat által.

A gépi tanulás és a mélytanulás alkalmazásai alapvetően lehetővé teszik, hogy a gesztusokat felismerjük és értelmezzük. A mély neurális hálózatok, mint a Convolutional Neural Networks (CNNs), lehetővé teszik a gesztusok felismerését képekből vagy videókból [15], ahogy a 2.2. ábrán is látszik.

A konvolúciós neurális hálózatok (CNN-ek) képfeldolgozásra optimalizált hálózatok, amelyek kiválóan alkalmazhatók gesztusok osztályozására, például ujjpozíciók vagy kéztartások felismerésére [16].



2.2. ábra. Konvolúciós neurális hálózat (CNN)

Forrás: www.medium.com

A konvolúciós neurális hálózat felépítése, ahogy a 2.2-es ábrán is látszik:

- **Konvolúciós réteg (Convolutional Layer)** - Ez a réteg kis méretű szűrőket (filterek) alkalmaz a bemeneti képen, amelyek végigcsúsznak a képen, és kis területek jellemzőit emelik ki (pl. ujjkontúr). A szűrők általában megtanulják, hogy milyen mintázatok fontosak a gesztusok felismeréséhez.
- **Aktivációs függvény (ReLU)** - Minden konvolúciós réteg után nemlinearitást vezetünk be, leggyakrabban ReLU (Rectified Linear Unit) segítségével. Ez növeli a hálózat képességét a bonyolult mintázatok megtanulására.
- **Pooling réteg** - Csökkenti a térbeli dimenziót azáltal, hogy a képből csak a legfontosabb jellemzőket tartja meg. Ez gyorsabbá és általánosabbá teszi a tanulási folyamatot.
- **Fully connected réteg (Teljesen összekapcsolt réteg)** - A kimeneti rétegek, ahol a rendszer végül döntést hoz (pl. melyik gesztusról van szó). Ezek a rétegek a konvolúciós rétegek által kivont jellemzőket használják az osztályozáshoz.
- **Kimeneti réteg (pl. Softmax)** - Olyan aktivációs függvény, amely valószínűségi eloszlást ad a gesztusosztályok között (pl. 90% eséllyel „mutatás”).

A CNN-ek hatalmas előnye, hogy képesek automatikusan kivonatolni a fontos jellemzőket anélkül, hogy manuálisan kellene definiálni őket. Ez különösen előnyös

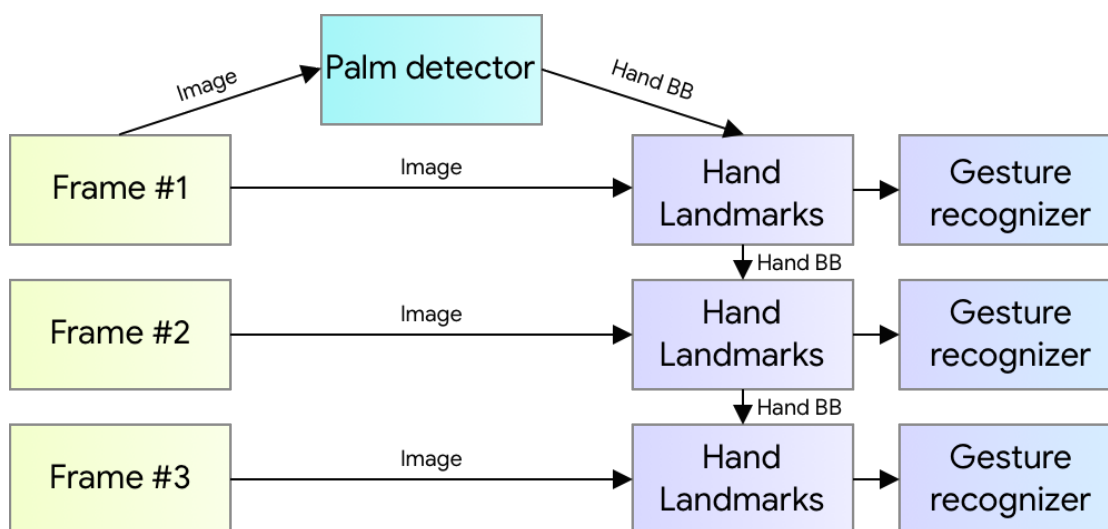
a gesztusfelismerésnél, ahol a mozdulatok sokfélesége miatt nagyon nehéz lenne explicit szabályokat megadni.

2.3. A MediaPipe működése és kézcsumópontok detektálása

A MediaPipe **Hands** modulja egy valós idejű kéz- és ujjpozíció-felismerő rendszer, amelyet a Google fejlesztett ki. Célja, hogy gyors és pontos módon ismerje fel a kéz 21 anatómiai pontját (landmarkjait, lásd 2.1. ábra) egyetlen képkockán belül, akár több kéz esetén is. A rendszer mögött egy többlépcsős, mesterséges intelligencián alapuló architektúra áll, amely optimalizálva lett CPU-n, GPU-n és mobil eszközökön való futtatásra is [17]. A felismerési folyamat fő lépései, ahogy a 2.3. ábrán is látható:

1. **Palm Detection – tenyérdetektálás:** A feldolgozás első lépése egy mély neurális hálózat (a BlazePalm modell), amely SSD (Single Shot Detector) alapú módszert alkalmazva egy *bounding box* segítségével lokalizálja a képen található tenyereket. Ennek célja nem a teljes kéz detektálása, hanem a gyors és robusztus inicializáció, hiszen a tenyér jól definiálható alakzat, ami stabil kiindulópontként szolgál.
2. **ROI kivágás és normalizálás:** A detektált tenyér alapján a rendszer kivág egy képrészletet (ROI - region of interest), amit egységes méretűre normalizál és orientál, hogy a következő lépés pontosabban működjön.
3. **Hand Landmark Model – kézlandmarkok regressziója:** Ez a második mélytanulási modell egy CNN (Convolutional Neural Network), amely képes 21 háromdimenziós (x, y, z) pontot meghatározni a kézen belül. Ezek a pontok az ujjvégeket, ízületeket és a csuklót reprezentálják. A modell a teljes kéz struktúráját egyszerre látja és értékeli, ezáltal képes értelmezni a kéz pozícióját és ujjainak hajlását is. A koordináták normált képtérben értendők, amelyet később a MediaPipe a kamera valós képméretéhez igazít.
4. **Kézkövetés (tracking):** Ha a kezdeti kézdetektálás sikeres volt, a következő képkockákon a rendszer már nem futtatja újra a drágább Palm Detection modellt, hanem a már felismert landmark-pozíciók alapján követi a kéz mozgását. Ez jelentősen csökkenti a számítási költséget és növeli a sebességet.

A **Hands** modul egyszerre akár több kéz követésére is alkalmas, és képes osztályozni, hogy a bal vagy a jobb kézről van szó. A rendszer a 21 pont 3D koordinátáját is képes kiszámítani, ahol a **z** érték relatív mélységet jelöl (a kamera síkjához viszonyítva).



2.3. ábra. MediaPipe Palm Detection működése

Forrás: www.ai.google.dev

A MediaPipe modelljeit több nyílt forrású keretrendszerből (pl. TensorFlow Lite) is lehet használni, és kifejezetten optimalizálva vannak WebAssembly, Android és iOS környezetekre, így lehetővé teszik a valós idejű, platformfüggetlen gesztusvezérlést [18, 19].

2.4. Hasonló alkalmazások és fejlesztések

A gesztusvezérlés széleskörű alkalmazást nyert különböző iparágakban, és számos projekt és kutatás foglalkozik ezzel a témával. Ezek az alkalmazások segítik a mindennapi életünket, azzal hogy nem kell egy fizikai eszközzel a kezünkben irányítsunk egy rendszert.

Okosotthonok

A gesztusvezérlés egyik legismertebb alkalmazása az okosotthon rendszerekben található. Az olyan alkalmazások, mint például a Google Home, Amazon Alexa, és az Apple HomeKit, bár hangalapú vezérlést használnak, az okosotthonok gesztusvezérlésére is egyre nagyobb figyelem irányul. A gesztusvezérlés növeli a használhatóságot, mivel lehetővé teszi az irányítást anélkül, hogy megérintenénk a vezérlő eszközöket [20]. Akár a mozgássérülteknek is megkönnyítené a mindennapi életüket ha gesztusokkal vezérelhetnék az otthonukat.

Autóipar

Az autókban a gesztusvezérlés alkalmazása gyorsan növekvő trend. A BMW és Audi már alkalmaznak gesztusvezérlő rendszereket a multimédia rendszerek, valamint egyéb autós funkciók irányítására. Ez növeli a vezetés biztonságát, mivel nem kell az autó multimédiás kijelzőjére figyelni vagy valamire kattintani rajta [21].

Egészségügy

A gesztusvezérlés steril környezetekben való alkalmazása is előtérbe került. A műtőkben például a gesztusvezérlés lehetővé teszi a sebészek számára, hogy érintés nélkül irányítsák a különböző orvosi berendezéseket. Az ilyen típusú alkalmazások jelentősen csökkenthetik a fertőzés kockázatát [22].

Művészet

A gesztusvezérlés a művészet és az interaktív installációk világában is egyre elterjedtebb. Egy angol énekes-dalszerző, zeneszerző és technológiai újító, Imogen Heap és egy fejlesztőcsapat közösen megalkották a MiMU Gloves-t. Ez egy fejlett gesztusvezérelt kesztyű, ami lehetővé teszi a zene érintés nélküli előadását. A kesztyűk érzékelik a kéz és az ujjak mozgását. Ezeket a mozdulatokat valós időben lefordítják MIDI jelekre vagy effektparancsokra. A művész így „a levegőben” irányíthatja a zenét, például hangerőt növelhet, visszhangot adhat egy szólamra, vagy új hangszert hívhat elő. Mindezt valós időben, késés nélkül, élő koncerten (NPR Music Tiny Desk Concert - 2019) is használható módon. Nem csak a zenében, hanem a festészetben is van alkalmazása a gesztusvezérlésnek, Leap Motion szenzor segítségével művészek festhetnek „a levegőben”, ahol a kézmozdulatokat 3D térben vizualizálják. A játékfejlesztés az egyik legkorábbi terület, ahol a gesztusvezérlés megjelent. Az olyan rendszerek, mint a Nintendo Wii, Microsoft Kinect vagy a PlayStation Move már évekkel ezelőtt is lehetővé tették, hogy a játékosok a testükkel irányítsák a karaktereket.

Oktatás

Dolgozatom során leginkább erre a felhasználására térek ki a gesztusvezérlésnek. Főként a digitális tananyagok interaktívabbá tételében játszik nagy szerepet. Egyre nagyobb kihívást jelent a diákok hosszú távú lekötése. Az érintésmentes, gesztusalapú vezérlés lehetővé teszi, hogy a tanárok és diákok természetes mozdulatokkal kommunikáljanak a digitális platformokon, ezáltal növelve az élményszerű tanulást [23]. A COVID-19 világjárvány idején jelentősen megnőtt az online és hibrid oktatási formák iránti igény, amely új típusú interakciós módszereket helyezett előtérbe.

- Virtuális osztálytermek: a valós idejű kézmozdulat-felismerés révén a diákok képesek válaszokat adni, jelezni, vagy navigálni a tananyagban anélkül, hogy fizikai beviteli eszközöket (egert vagy billentyűzetet) kellene használniuk. Az olyan rendszerek, mint a Microsoft Kinect vagy a Leap Motion, korai példái voltak az ilyen típusú interakciónak. A tanulmányok szerint az ilyen technológiák alkalmazása elősegíti a tanulók aktívabb részvételét és fokozza a figyelmet a tanórák során [24].
- Különleges nevelési igényű tanulók támogatása: a mozgáskorlátozott tanulók számára lehetőséget biztosít arra, hogy kéz- vagy testmozdulatokkal vezéreljenek oktatási szoftvereket, ami hozzájárul az esélyegyenlőség megteremtéséhez. Egy 2020-as tanulmány kimutatta, hogy a testreszabható gesztusalapú felületek jelentősen javíthatják a mozgáskorlátozott diákok digitális tanulási élményét [25].
- Élményalapú tanulás: a gesztusvezérlés különösen hatékony eszköz lehet olyan tanulási környezetekben, ahol a tananyag interaktív 3D modellek vagy szimulációk formájában jelenik meg. Például virtuális laboratóriumokban vagy biológiaórákon a tanulók kézmozdulatokkal forgathatnak, manipulálhatnak vagy felnagyíthatnak különböző objektumokat, ezzel elősegítve a fogalmak jobb megértését és a tanulás mélyebb élményét [26].

3. fejezet

Tervezés

3.1. Célkitűzések

A kutatás célja volt egy olyan gesztusfelismerő API, amit széles körben és egyszerűen lehet használni. Fontosnak tartottam, hogy továbbfejleszthető és könnyen bővíthető legyen, illetve integrálható legyen más rendszerekben is.

Dolgozatom során három alkalmazási területet érintettem. Az egyik az oktatást segítő és interaktívabbá tevő kliens alkalmazás. Ezzel az alkalmazással megkönnyíteném az oktatók feladatát úgy, hogy ne kelljen egeret vagy billentyűzetet használniuk az online prezentálás során, hanem a kéz gesztusaikkal vezérelhetik azt. Például egy „balra lapozás” gesztussal tovább lapozhassák a bemutató oldalait. Az oktató beállíthatja hogy rajta akar-e lenni a prezentáció oldalain, ilyenkor a kamera képét körbevágjuk, hogy csak a tanár látszódjon, a háttér ne, és rátesszük a bemutatóra. Így interaktívabbá és jobban követhetővé teszi az online tanórát. Ilyenkor például egy „nagyítás” gesztussal kinagyíthatja magát a felhasználó a képen. Ezek mellett még öt opciót be tud állítani, vagy a gesztusaival vezérelhet, például: teljes képernyős mód, debug mód, ahol megjeleníti a kéz csomópontokat és vizualizálja hogy a számítógép hogy ismeri fel a felhasználó kezét, gesztusait. Emellett a felhasználó személyre szabhatja a beállítások között, hogy melyik gesztust milyen funkció kövessen.

A másik alkalmazási terület amit érintettem, az az okosotthonok. A kutatásnak célja volt az is, hogy ne csak egy felhasználást mutassak be, így készítettem egy okos lámpát, amit kéz gesztusokkal lehet fel- illetve lekapcsolni. Számtalan hangvezérléssel működő okosotthon rendszer létezik már, de fontosnak találom, hogy ne csak a hangunkkal, hanem a mozdulatainkkal is tudjuk irányítani otthoni berendezéseinket, mert vannak olyan helyzetek amikor nem előnyös a beszéd, például ha egy kisgyerek alszik, akkor ne ébresszük fel amiatt, hogy bekapcsolnánk a klímát az egyik szobában.

A harmadik terület egy egyszerűbb alkalmazása a gesztusvezérlésnek. Mobiltelefonra kiküld egy adott értesítést a felismert gesztus után. Ezzel az a cél, hogy bemutattam, hogy mobiltelefonos felhasználásra is alkalmas és ilyen irányban is felhasználható az API-m. Például: egy felismert gesztus után értesítés kapok a telefonomra, hogy valami történt, egy email kiküldésre került, vagy akár egy képet is készíthetek így a telefonommal.

3.2. Felhasználói igények

Az API sok fejlesztő számára nyújt lehetőséget, olyan alkalmazást fejleszteni, ami valós időben detekál gesztusokat, amit majd ők felhasználhatnak saját igényeik szerint.

Az oktatást segítő alkalmazásom célcsoportja a tanárok és diákok, leginkább online vagy hibrid oktatási környezetben. Arra a problémára ad megoldást, hogy az oktatók kényelmesebben és tesközelibb módon vezessék le a tanórkát, míg a diákok figyelmét jobban fenntartják azzal, hogy az oktató látszódik a bemutatón és úgy mutogat, magyaráz.

Az okosotthon alkalmazása elsősorban olyan felhasználóknak nagy segítség, akik mozgáskorlátozottak, például: ágyban fekvő betegek. Megkönnyíti számukra a lakás elektronikai eszközeinek használatát. De nem csak nekik nagy segítség, hanem bárki számára, akik szeretik a kényelmet és a technológiai újításokat a lakásában.

3.3. Rendszer követelményei

Funkcionális követelmények

- Az API képes legyen különböző kézmozdulatok felismerésére.
- Az API alkalmazható legyen különböző környezetekben (web, IoT, mobil).
- A felhasználó tudja kiválasztani, hogy egy adott gesztus mit vezéreljen (pl. lapozás, nagyítás).
- A lámpa gesztusra kapcsoljon be/ki.
- Az API küldjön értesítést a telefonra, ha egy adott gesztust észlel.

Nem funkcionális követelmények

- A kézmozdulat-felismerés gyors és pontos legyen.
- A gesztusfelismerésnek valós időben kell történnie.

- Alacsony erőforrásigény, fusson laptopon GPU nélkül is.
- Az API dokumentált és könnyen integrálható legyen külső rendszerekbe.
- A webes alkalmazás platformfüggetlenül működjön, modern böngészőkön keresztül, telepítés nélkül.
- Legyen felhasználóbarát az oktatást segítő felület.

3.4. Tervezési alapelvek és architekturális megfontolások

A rendszer megtervezésének kiindulópontját a funkcionális és nem funkcionális követelmények jelentették, ugyanakkor a cél nem csupán egy működőképes prototípus létrehozása, hanem egy olyan tudományosan is releváns megközelítés kialakítása volt, amely hozzájárulhat az interaktív ember-gép kapcsolatok fejlődéséhez. A gesztusvezérlés témájában végzett kutatások [23, 25, 20] rámutattak arra, hogy a felhasználói élmény és a technikai hatékonyság közötti egyensúly megteremtése kulcsfontosságú a széles körű adaptáció szempontjából. Ezért a tervezési döntések az alábbi alapelvek mentén születtek:

- **Valós idejűség:** A szakirodalom alapján [9] a felhasználói élmény egyik kritikus tényezője a késleltetés minimalizálása. A MediaPipe keretrendszer kiválasztása lehetővé teszi a kézmozdulatok gyors, valós idejű felismerését akár CPU-alapú környezetben is, mivel a modell optimalizált és alacsony számítási igényű.
- **Erőforrás-hatékonyság:** Az alacsony erőforrásigény nemcsak technikai szükségyszerűség (pl. laptopon, GPU nélkül történő működés), hanem fontos tudományos célkitűzés is, hiszen csak így válhat a rendszer széles körben alkalmazhatóvá, akár oktatási vagy otthoni környezetben is [24]. Ezért olyan architektúra került kialakításra, amely elválasztja a gépi tanulás alapú gesztusfelismerést az alkalmazáslogikától, lehetővé téve a különböző platformokon való futtatást.
- **Modularitás és újrafelhasználhatóság:** A rendszer komponensei (pl. kézfelismerő modul, eseménykezelő API, vezérlő interfész) egymástól függetlenül cserélhetők vagy fejleszthetők. Ez az architekturális döntés lehetővé teszi, hogy a későbbiekben különböző oktatási forgatókönyvekhez, tanulói profilokhoz vagy fizikai eszközökhöz is könnyedén adaptálható legyen a rendszer.
- **Platformfüggetlenség és skálázhatóság:** A webes technológiák (pl. JavaScript, WebSocket) alkalmazása biztosítja, hogy a rendszer böngészőből

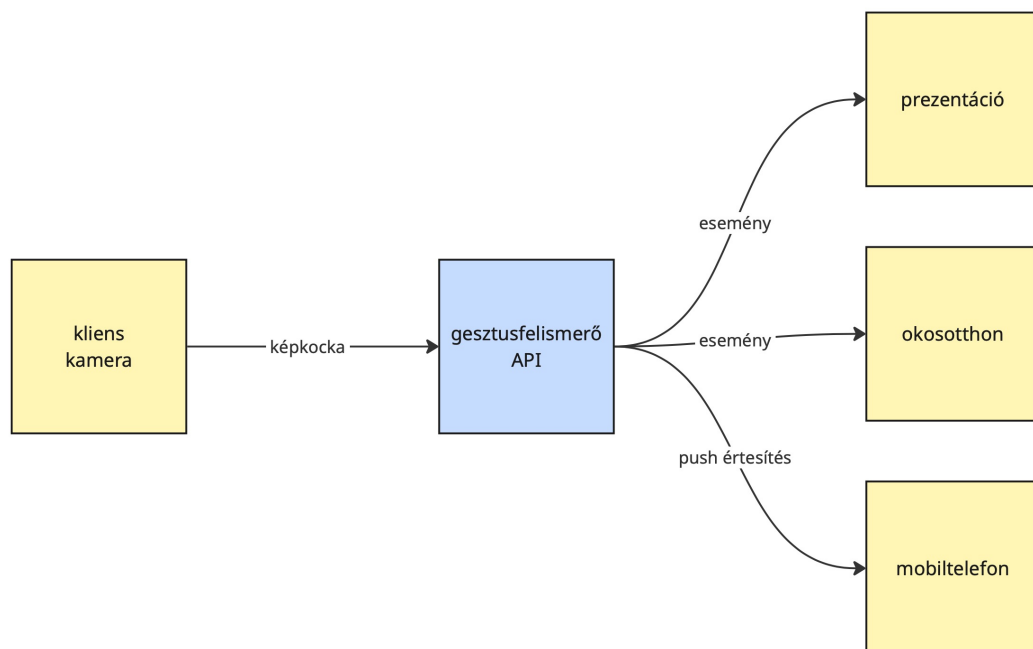
is elérhető legyen, telepítés nélkül, így széles körben tesztelhető és demonstrálható például iskolai környezetben is [26]. Az API alapú kommunikációs struktúra pedig lehetővé teszi az IoT-eszközökkel való integrációt.

- **Interdiszciplináris célkitűzések támogatása:** A tervezés során figyelembe vételre került, hogy a gesztusvezérlés nem csak technológiai, hanem pedagógiai és társadalmi szempontból is hozzáadott értéket képvisel. Az oktatási felhasználási esetek és a mozgáskorlátozott felhasználókra való tekintettel olyan nyílt, testreszabható architektúra került kidolgozásra, amely képes különböző tanulási helyzetekhez alkalmazkodni.

4. fejezet

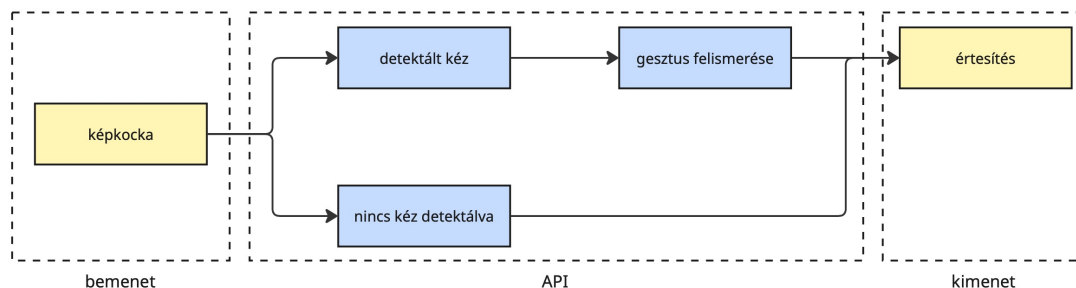
Rendszer bemutatása

A rendszer több komponensből áll, egy API és a kliens alkalmazások. Az API bemeneként képkockákat vár, majd kimenetként eseményeket küld ki a kliens alkalmazásoknak, ahogy a 4.1. ábrán is látható.



4.1. ábra. A rendszer működése

4.1. Gesztusfelismerő API



4.2. ábra. Az API folyamatábrája

Az API egy szerveren futtat, ahol elvégzi a kézdetektálást, gesztusfelismerést és értesítések kiküldését a feliratkozott klienseknek. A 4.2. ábrán látható, hogy az API fogadja a kliensől a képkockákat egy endpointon, majd detektálja, hogy van-e rajta kéz, vagy nincs. A kéz detektálása úgy történik, hogy a modell képes felismerni a kéz csomópontjait (lásd 2.1-es ábra) majd visszaadni azt, így el lehet dönteni, hogy van-e felismerendő kéz gesztus a képen. Ha nincs felismert kéz, akkor „no hand detected” értesítést küld ki, ha van kéz, akkor a következő lépés a gesztus felismerése. Lehetnek statikus (például „option 1”) és dinamikus (például „swipe right”) gesztusok.

A statikus gesztusok felismerése az egyszerűbb, mivel azok csak kéztartások, nem pedig egy mozdulat. Úgy ismeri fel ezeket a rendszer, hogy a felismert csomópontok alapján a rendszer szöget számít két pont között. Ez azért előnyösebb mint a távolság kiszámítása kép csomópont között, mert így nem számít, hogy a kéz milyen távolságra van a kamerától. Ha így csinálnánk és be lenne állítva hogy a „pointing” (mutatóujj fel, többi ujj összeszorítva) esetében a mutatóujj és a csukló közti távolság 120, akkor ha távolabb van a kezünk a kamerától, akkor nem ismeri fel a gesztust. Szög számítása esetén elkerüljük ezt a problémát. Például a „zoom out” (összecsípés) esetén kiszámítjuk a következő szögeket:

- THUMB_TIP – WRIST – INDEX_FINGER_TIP
- INDEX_FINGER_TIP – WRIST – MIDDLE_FINGER_TIP
- MIDDLE_FINGER_TIP – WRIST – RING_FINGER_TIP
- RING_FINGER_TIP – WRIST – PINKY_TIP

A küszöbértékeket ismételt tesztelés és megfigyelés alapján határoztam meg. A gesztusok felismeréséhez szükséges szögek nem egzakt módon rögzítettek, mivel

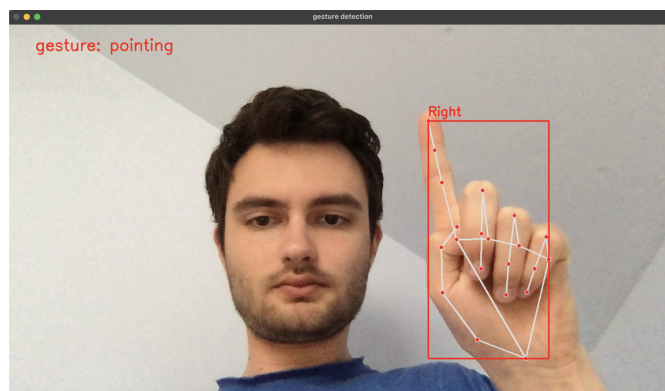
azok függenek a kéz méretétől, tartásától, valamint a kamera szögétől és pozíciójától is. Ennek megfelelően több felhasználóval és különböző kéztartásokkal végzett próba során figyeltem meg, hogy mekkora szögek esetén jellemző az adott gesztus.

Konkréten a „Zoom Out” (összecsípés) gesztushoz tartozó szögek vizsgálata során azt tapasztaltam, hogy ha a hüvelykujj és kisujj közötti ujjpercekre számított szögek mindegyike egy adott küszöbérték (pl. $\theta = 4^\circ$) alá esik, akkor az ujjbegyek valóban egymáshoz közel kerülnek. Ezt a küszöböt iteratív módon állítottam be: kezdetben magasabb értékről indultam, majd fokozatosan csökkentettem, amíg el nem értem azt a pontot, ahol már biztosan csak a kívánt mozdulatokra történt felismerés, de a rendszer még nem vált túl érzéketlenné.

A végső küszöbérték kiválasztása tehát egy egyensúlyi pont keresése volt: elég szigorúnak kellett lennie a téves felismerések elkerüléséhez, de elég megengedőnek ahhoz, hogy a különböző felhasználók kézmozdulatai is megfeleljenek neki. Ezt a küszöbértéket szükség esetén dinamikusan is lehet hangolni a jövőben, felhasználói visszajelzések vagy további gépi tanulás alkalmazásával.

A dinamikus gesztusok egy mozgást írnak le, például „swipe left” (balra húzás). Ebben az esetben a kéz detektálás után kiszámítja az összes kézcsomópontpont (landmark) X-koordinátájának átlagát. Ez az érték reprezentálja a kéz vízszintes helyzetét a képernyőn. Lementi az előző pozíciókat, hogy legyen mihez hasonlítani a mozgás felismerés miatt. Ha az új pozíció jelentősen balrább van (legalább 0.05-tel kisebb), akkor „swipe left” (balra húzás) gesztust ismeri fel. Majd a jelenlegi pozíciót eltárolja következő összehasonlításhoz. Itt is, mint a statikus gesztusok esetén, miután megtörtént a felismerés, utána kiküldi a gesztus nevét a kliensnek.

Van egy opcionálisan megjelenő debug mód is, ami a 4.3. ábrán látható, ahol az alkalmazás kirajzolja a képernyőre a detektált kezdet (kézcsomópontokat). Ez azért volt fontos, hogy fejlesztés során lássam, hogy melyik pontokból és milyen szögeket számoljak ki.



4.3. ábra. Kézcsomópontok megjelenítése (debug mód)

Hibakezelés és megbízhatóság

A rendszer alapvető hibakezelési mechanizmusokkal van ellátva annak érdekében, hogy a felhasználói élmény még váratlan körülmények esetén is fenntartható maradjon. Amennyiben a kamera nem érhető el (pl. le van tiltva vagy fizikailag nincs csatlakoztatva), a kliensoldali alkalmazás erről egy figyelmeztető üzenetben tájékoztatja a felhasználót, és automatikusan leállítja a képfeldolgozást. Ebben az esetben nem működik a *prezentáció* funkció. Ha az API nem válaszol például hálózati hiba vagy szerverleállás miatt, akkor ugyan úgy nem érhető el a *prezentáció* funkció.

A MediaPipe könyvtár esetén is előfordulhat, hogy egy adott képkockán nem sikerül kezdetet detektálni. Ilyenkor a rendszer „no hand detected” üzenetet küld ki a klienseknek.

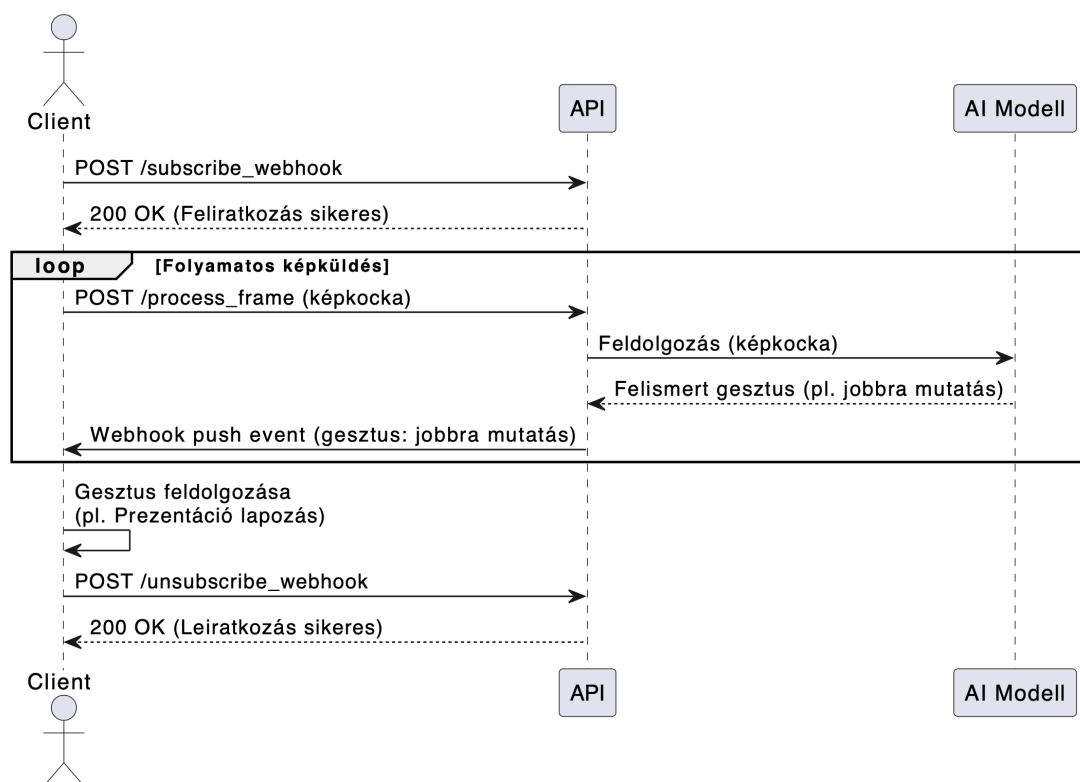
4.2. Kliens alkalmazások

4.2.1. Oktatást segítő prezentációs alkalmazás

Ez egy böngészőben elérhető, platformfüggetlen webalkalmazás, amely a gesztusvezérlés lehetőségeit kihasználva segíti az oktatók és előadók munkáját. Az alkalmazás négy fő oldalt tartalmaz:

- **Home** – kezdőoldal, rövid leírással a rendszer működéséről, céljáról és használatáról.
- **Presentation** – a prezentációk vetítésére szolgáló felület, ahol gesztusokkal lehet lapozni a diák között.
- **Practice** – gyakorlófelület, ahol a felhasználó kipróbálhatja és ellenőrizheti, hogy a gesztusait helyesen értelmezi-e a rendszer.
- **Settings** – beállítások oldala, ahol engedélyezhető például a debug mód, illetve testreszabható, hogy mely gesztus milyen műveletet váltson ki.

A **Presentation** nézetben a felhasználó feltöltheti a bemutatni kívánt prezentációt, majd a saját eszközének kameráját használhatja gesztusvezérlésre: például diaváltásra („swipe left” / „swipe right”), nagyításra („zoom in/out”) vagy opcióválasztásra („option 1/2/3”). A 4.4. ábrán látható időrendben egy prezentáció bemutatásának folyamata.



4.4. ábra. Szekvencia diagram

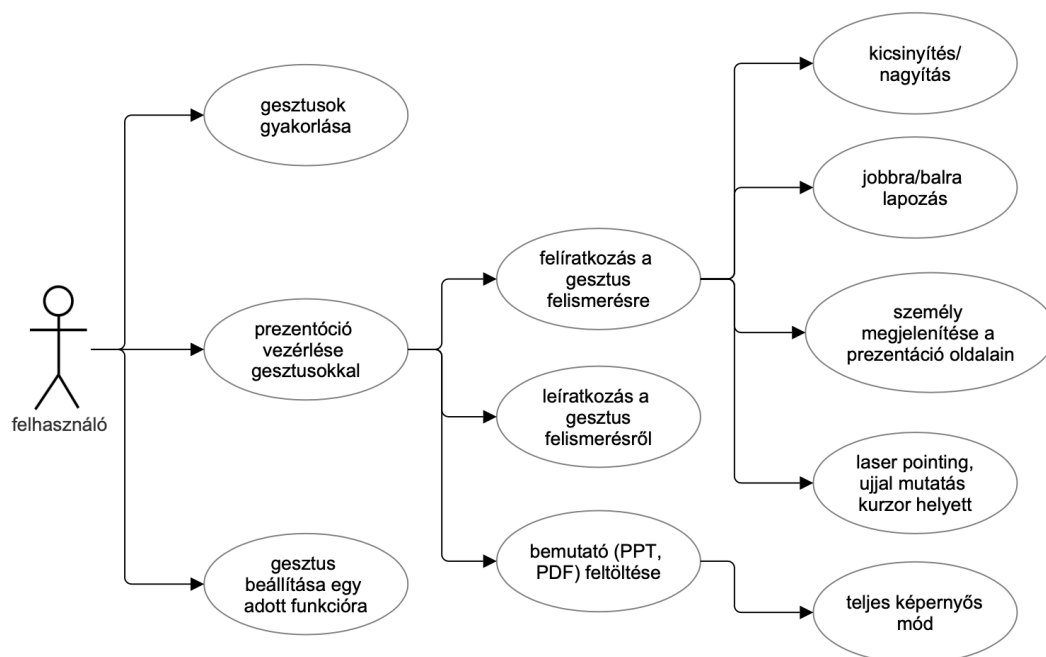
A **Practice** felület a felhasználónak lehetőséget biztosít arra, hogy, a gesztusokat előre begyakorolja és tesztelje azok felismerését. A detektált kézcsomópontokat és az éppen értelmezett gesztust valós időben jeleníti meg. Itt debug módban vizuálisan láthatóak a csomópontok, amelyek a gesztusdetektálás alapját képezik.

Az alkalmazás regisztráció nélkül is használható, ezzel is támogatva az egyszerű és gyors kipróbálást oktatási környezetben.

Az alábbi 4.5. ábra bemutatja az alkalmazás funkcióit és a felhasználókkal való kapcsolatát.

Az alkalmazás látványos funkciója a „személy megjelenítése”, amely mesterséges intelligencián alapuló modellre épül.

A **személy megjelenítése** funkció során valós idejű szegmentálás történik a be-meneti kameraképen. A rendszer az AI modell segítségével elkülöníti a felhasználót a háttértől, körbevágja a személy alakját, majd átlátszó háttérrel megjeleníti őt a prezentációk diáin. Ez vizuálisan azt az élményt kelti, mintha az előadó közvetlenül a bemutatóban szerepelne, a diák előtt állva.



4.5. ábra. Use Case diagram

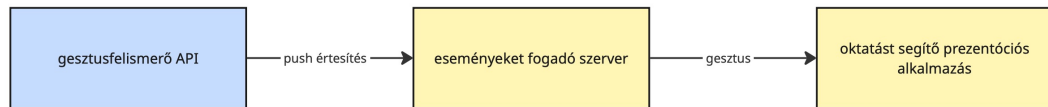
Fontos megjegyezni, hogy ez a szegmentálás teljes egészében kliensoldalon történik. A kiválasztott AI modell kellően alkalmas ahhoz, hogy böngészőben is futtatható legyen, külső számítási erőforrás nélkül. Ez nemcsak csökkenti a hálózati terhelést, hanem jelentősen gyorsítja az interakciókat is, mivel a képkockák nem kerülnek vissza az API szerverre feldolgozásra, így csak a gesztusfelismerés eredménye, az adott gesztus neve továbbítódik a szerver és a kliens között.

Ez a megoldás nemcsak hatékonyabb, hanem architektúráisan is indokolt, mivel az API oldalán a képszegmentálás eredménye más funkcióhoz nem szükséges, így a feldolgozást felesleges lenne ott végezni. A feldolgozás kliensoldali megvalósítása ezzel erőforrás-takarékosabb.

Webhook szerver működése

A kliensalkalmazás és az API közötti kommunikációt egy különálló szerver végzi, ahogy a 4.6. ábrán is látható, amely webhook és websocket technológiákon keresztül továbbítja az eseményeket. Az API minden gesztusfelismerés után egy HTTP POST kérést küld a szerver végpontjára. Ez a szerver továbbítja az érkező adatokat a kliens felé websocketen keresztül, így az alkalmazás valós időben tud reagálni a gesztusokra. A webhook szerver és websocket kapcsolat előnye, hogy nem szükséges

folyamatosan lekérdezni az API-t, mivel a gesztusfelismerés eseményvezérelten működik. Ez nemcsak hatékonyabb, hanem a felhasználói élményt is javítja a késleltetés minimalizálásával.



4.6. ábra. API - kliens kapcsolat

Az API a következő formátumban küldi a gesztus eseményeket a webhook szervernek:

```
1 POST /webhook
2 Content-Type: application/json
3 {
4   "gesture": "Zoom In"
5 }
```

A webhook szerver Express.js segítségével fogadja ezt a POST kérést, majd továbbítja az információt websocket kapcsolaton keresztül a React frontendnek:

```
1 app.post("/webhook", (req, res) => {
2   console.log("Received Webhook:", req.body);
3   io.emit("gesture_event", req.body);
4   res.json({ status: "received" });
5 });
```

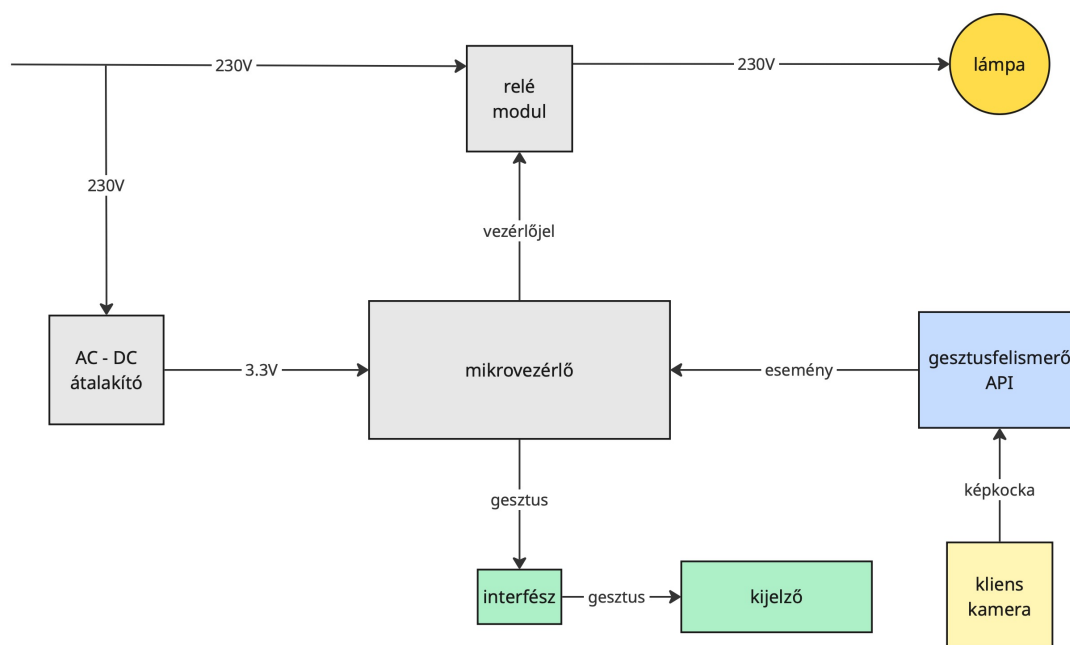
A React alkalmazás pedig az alábbi módon figyeli a websocket eseményeket, és frissíti a megjelenített gesztust:

```
1 useEffect(() => {
2   socket.on("gesture_event", (data) => {
3     console.log("Gesture received:", data);
4     setGesture(data.gesture);
5   });
6   return () => {
7     socket.off("gesture_event");
8   };
9 }, []);
```

Ez a struktúra biztosítja, hogy a felhasználó gesztusai az API-tól egészen a kliensalkalmazásig egységes formátumban és valós időben továbbításra kerüljenek

4.2.2. Gesztusvezérelt okos lámpa

Egy elektromos hálózatban levő lámpa vezérlése gesztusokkal. A 4.7. ábrán látható, hogy egy kamera veszi a képet, a képkockákat továbbítja az API-nak. A mikrovezérlőt feliratkoztatjuk a gesztus felismerésre, így onnantól kezdve vezeték nélküli értesítést kap minden felismert gesztus után. A vezérlésre (lámpa le- illetve felkapcsolására) beállított gesztus felismerése után jelt küld a relének, ami ebben az esetben egy kapcsolóként szolgál. Opcionálisan megjeleníthetjük a felismert gesztus nevét egy kijezőn, a jobb átláthatóság érdekében. A mikrovezérlőnek is áramforrásra van szüksége, így egy váltóáram - egyenáram átalakítóval kap tápot.

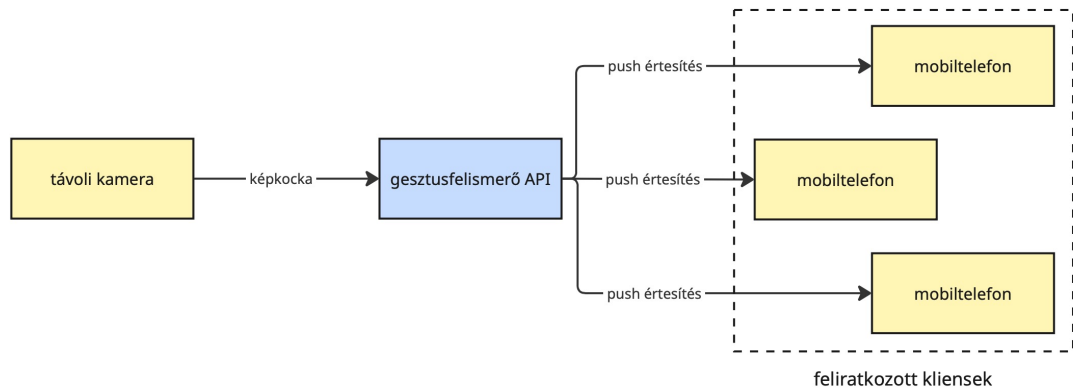


4.7. ábra. Gesztusvezérelt lámpa architektúrája

4.2.3. Push értesítések mobil eszközre

A rendszer képes arra, hogy a gesztusfelismerés eredményeit valós időben eljuttassa külső eszközökhöz, például mobilalkalmazásokhoz push értesítések formájában (lásd 4.8. ábra). Ez különösen hasznos olyan esetekben, amikor a felhasználó nincs közvetlen kapcsolatban a vezérlőfelülettel, de szeretne értesülni a rendszer

által észlelt eseményekről. Az API szerver a beérkező képkockák alapján elvégzi a gesztusfelismerést. A felismerés után a rendszer egy előzetesen regisztrált endpointokra (webhook URL-ekre) küld HTTP POST kérést, amely tartalmazza a felismert gesztus nevét. Ezeket az endpointokat a kliensalkalmazások – például mobil eszközökön futó alkalmazások – előzetesen feliratkozással regisztrálhatják az API-nál.



4.8. ábra. A feliratkozott kliensek értesítést kapnak gesztus felismerés után

5. fejezet

Megvalósítás

5.1. Gesztusfelismerő API

A backend rendszer Python nyelvben készült, és a **FastAPI** keretrendszert használja. A FastAPI modern, aszinkron REST API-k gyors fejlesztésére alkalmas.

A FastAPI választása több szempontból is indokolt. Mivel a gesztusfelismerés valós idejű képfeldolgozást igényel, elengedhetetlen a gyors válaszidő és a jó skálázhatóság. A FastAPI a Python `asyncio` moduljára épülve lehetővé teszi az aszinkron működést, így a szerver egyszerre több kérést is képes hatékonyan kezelni anélkül, hogy blokkolná a folyamatokat. Ez a tulajdonság különösen előnyös olyan alkalmazásoknál, ahol a válaszidő kulcsfontosságú.

Emellett a FastAPI modern fejlesztői élményt nyújt: automatikusan generál Swagger és OpenAPI dokumentációt a meglévő kód alapján, amely megkönnyíti a fejlesztők és tesztelők munkáját, különösen a kliensoldali integrációk során. A keretrendszer támogatja a Python típusannotációit, melyeknek köszönhetően már fejlesztés közben is ellenőrizhető a bemeneti adatok helyessége. A FastAPI szorosan integrálódik a `Pydantic` könyvtárral, amely automatikusan végzi az adatok validálását és konvertálását, ezáltal csökkentve a hibák számát.

A rendszer fő célja, hogy feldolgozza a kientől érkező képkockákat, felismerje a kézmozdulatokat, és az eredményt továbbítsa a webhookokra feliratkozott kliensek számára.

5.1.1. Webhook kezelés

A webhookok push alapú értesítési módszerként különösen hasznosak a gesztusfelismeréshez, mivel számos előnyt kínálnak a rendszer számára. Az egyik legnagyobb előnyük a valós idejű értesítések biztosítása. Mivel a webhookok lehetővé teszik, hogy az API közvetlenül értesítse a feliratkozott klienseket, azonnali reakcióra van

lehetőség. Ez különösen fontos a gesztusfelismerésnél, ahol az időzítés kulcsfontosságú a felhasználói élmény szempontjából.

A webhookok aszinkron módon működnek, így nem blokkolják az alkalmazás működését. Ez lehetővé teszi a rendszer számára, hogy párhuzamosan, gyorsan és hatékonyan kezelje a bejövő gesztusokat és értesítéseket, anélkül hogy túlterhelné az API-t. Ez az aszinkron működés segít a rendszer optimalizálásában, mivel az értesítések feldolgozása nem igényel folyamatos kérdezést vagy szinkronizációt, amely lelassíthatná a folyamatot.

A webhookok skálázhatóságot is biztosítanak, mivel lehetővé teszik a rendszer számára, hogy több klienst is kiszolgáljon anélkül, hogy a központi logikát módosítani kellene. A webhookok alapvetően a feliratkozott URL-ek listájára küldenek értesítéseket, ami azt jelenti, hogy a rendszer könnyen bővíthető a jövőben, például új kliensek hozzáadásával, anélkül hogy átfogó változtatásokat kellene végrehajtani.

A webhookok emellett erőforrás-hatékonyak is, mivel csak akkor aktiválódnak, amikor egy esemény történik (például egy gesztus felismerése). Ez csökkenti az erőforrások folyamatos felhasználását, valamint minimalizálja a felesleges hálózati forgalmat, mivel nem kell rendszeresen lekérni az adatokat, hanem csak akkor történik adatküldés, amikor szükséges.

A webhookok alapértelmezés szerint HTTP POST kéréseket használnak, a külső rendszerek könnyen beállíthatják saját URL-jeiket a webhook fogadására, anélkül hogy bonyolult konfigurációs lépésekre lenne szükség.

Az alábbi két endpointja lehetővé teszi, hogy külső kliensek fel- és leiratkozzanak a gesztusértesítésekre. A feliratkozás egy URL-t vár, amelyre a rendszer POST kérést küld, ha új gesztust ismer fel.

```
1 @router.post("/subscribe_webhook")
2 async def subscribe_webhook(url: str):
3     if url in subscribers:
4         raise HTTPException(status_code=400, detail="Already
5         subscribed")
6         subscribers.add(url)
7     return {"message": f"Subscribed: {url}"}
```

A `/subscribe_webhook` endpoint elmenti a feliratkozó URL-címeket egy memóriában tárolt `subscribers` halmazba. Ezt követően minden új gesztus esetén ezekre az URL-ekre küldjük el az eredményt.

```

1 @router.post("/unsubscribe_webhook")
2 async def unsubscribe_webhook(url: str):
3     if url not in subscribers:
4         raise HTTPException(status_code=400, detail="Not
5         subscribed")
6     subscribers.remove(url)
7     return {"message": f"Unsubscribed: {url}"}

```

A leiratkozás hasonlóan működik – ha az adott URL szerepel a nyilvántartásban, akkor törlésre kerül.

5.1.2. Képkockák feldolgozása

A gesztusfelismerés kulcsa a kliens által feltöltött képkockák (frame-ek) feldolgozása. Ehhez a rendszer a `/process_frame` endpointot használja, amely fogad egy képfájlt, és az alapján meghatározza a gesztust.

```

1 @router.post("/process_frame")
2 async def process_frame(frame: UploadFile = File(...)):
3     img_array = np.frombuffer(await frame.read(), np.uint8)
4     img = cv2.imdecode(img_array, cv2.IMREAD_COLOR)
5     img = cv2.flip(img, 1)
6     gesture, _ = process_hand_gesture(img)

```

Itt a `cv2.flip` függvénnyel tükrözzük a képet, hogy a gesztusfelismerés megfeleljen a felhasználói nézőpontnak (selfie-mód). A `process_hand_gesture` függvény a képet továbbküldi a MediaPipe-alapú feldolgozónak.

5.1.3. Gesztus feldolgozása és továbbítása

Ha a felismerés sikeres, és a detektált gesztus megfelel bizonyos kritériumoknak (pl. összecsípés, nagyítás, lapozás), akkor a rendszer továbbítja azt a webhookokra:

```

1 await notify_subscribers(gesture)

```

Ez biztosítja, hogy csak megbízhatóan észlelt gesztusok kerüljenek továbbításra.

5.1.4. Gesztusfelismerés a MediaPipe segítségével

A gesztusfelismeréshez a *Google MediaPipe Hands* modult alkalmaztam, amelyet korábban a 2.3 alfejezetben részletesen ismertettem. A választás egyik fő indoka az

volt, hogy a MediaPipe olyan valós idejű kézdetektálást és követést tesz lehetővé, amely megfelel a 3.3 alfejezetben bemutatott teljesítmény- és pontossági követelményeknek: gyors válaszidő, mobil eszközökön való futtathatóság, valamint pontos 3D landmark predikció.

A rendszer képes egy vagy több kéz detektálására és 21 kulcspont (landmark) koordinátáinak meghatározására, ahogy a 2.1. ábrán is látható, amelyeket később a gesztusok osztályozásához használunk fel. A landmarkok segítségével nem csak a kéz formáját, hanem annak orientációját és mozgását is követhetjük.

A választott technológia a fejlesztési szakaszban teljesítette az alábbi követelményeket:

- A rendszer képes legyen kézmozdulatokat valós időben detektálni.
- A megoldás legyen platformfüggetlen, akár mobil eszközökön is használható.
- A kézcsomópontokból egyszerűen lehessen gesztusokat osztályozni.

Feldolgozási lépések

A `process_hand_gesture` függvény először RGB-re konvertálja a képet (a MediaPipe ezt várja bemenetként), majd lefuttatja a detekciós modellt.

```
1 def process_hand_gesture(frame):
2     frame_rgb = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)
3     hand_results = hand_gesture_detector.detect_hands(
        frame_rgb)
```

Ezután, ha található kéz a képen, a rendszer megpróbál gesztust azonosítani a `detect_gesture()` függvény segítségével.

Statikus gesztusok detektálása

A statikus gesztusok felismerése az ujjak egymáshoz és a csuklóhoz viszonyított szögének meghatározásán alapszik. Ehhez az alábbi függvény kiszámítja a két ujjhegy és a csukló által bezárt szöget.

```
1 def calculate_angle(p1, p2, point):
2     vector_1 = np.array([p1.x - point.x, p1.y - point.y])
3     vector_2 = np.array([p2.x - point.x, p2.y - point.y])
4     dot_product = np.dot(vector_1, vector_2)
5     magnitude_1 = np.linalg.norm(vector_1)
6     magnitude_2 = np.linalg.norm(vector_2)
```

```

7   if magnitude_1 == 0 or magnitude_2 == 0:
8       return 0
9   cos_theta = dot_product / (magnitude_1 * magnitude_2)
10  angle_rad = np.arccos(np.clip(cos_theta, -1.0, 1.0))
11  return np.degrees(angle_rad)

```

A lenti kódrészletben segítségével például detektálható a „zoom out” gesztus, ha az összes ujj szinte teljesen egymás mellett van.

```

1  if (thumb_index_angle < 4 and index_middle_angle < 4 and
2      middle_ring_angle < 5 and ring_pinky_angle < 4):
3      gesture = "zoom out"

```

Ez lehetővé teszi a finom kéztartásbeli különbségek megbízható felismerését.

Dinamikus gesztusok detektálása

A mozgás alapú gesztusok (pl. „swipe left” vagy „swipe right”) detektálása a kéz vízszintes irányú elmozdulásán alapszik, amit a rendszer képkockákon át követ. Ezek felismerése külön logikát igényel, és nem csak egyetlen képkockán múlik.

A dinamikus gesztusok, például a kéz jobbra vagy balra történő elmozdítása, képkockák közötti mozgás alapján kerülnek felismerésre. A detekció, a kéz vízszintes (X) koordinátájának nyomon követése.

A `detect_movement()` függvény kiszámítja a kéz X-koordinátáinak átlagát, és összehasonlítja az előző képkockában mért értékkel.

```

1  def detect_movement(self, hand_landmarks):
2      media_x = round(sum([lm.x for lm in hand_landmarks[0].
3                          landmark]) / len(hand_landmarks[0].landmark), 2)
4
5      if self.last_position is not None:
6          if media_x < self.last_position - 0.05:
7              movement = "Swipe Left"
8          elif media_x > self.last_position + 0.05:
9              movement = "Swipe Right"
10         else:
11             movement = "None"
12     self.last_position = media_x
13     return movement

```

Ez a logika egyszerűen és hatékonyan képes felismerni, ha a kéz határozottan elmozdul jobbra vagy balra.

Példa: swipe left

Tegyük fel, hogy egy felhasználó a kézfejét balra húzza. A kéz landmark pontjainak átlagos X-pozíciója jelentősen csökken:

- Előző pozíció: 0.6 - Aktuális pozíció: 0.5 → különbség: -0.1 → **swipe left** detektálva

Ugyanez fordított esetben **swipe right** mozgulatot eredményez.

Érvényes gesztus kiválasztása

Mivel a MediaPipe érzékeny lehet a zajra, a felismerést egy gesztustörténet alapján szűrjük. Csak akkor térünk vissza egy gesztussal, ha az legalább háromszor egymás után ugyanaz, ahogyan a lenti kódrészletben is látszik.

```
1 self.gesture_history.append(gesture)
2 if len(self.gesture_history) > 10:
3     self.gesture_history.pop(0)
4
5 if self.gesture_history.count(gesture) >= 3:
6     return gesture
```

Ez a módszer segít kiszűrni az esetleges téves vagy pillanatnyi mozgulatokat, amelyek nem reprezentálnak valós felhasználói szándékot. A gesztusérzékelés során gyakran előfordulhatnak rövid ideig tartó, félreértelmezhető kézpozíciók (pl. árnyék, elmozdulás, bemozdulás), amelyek nem kívánt eseményekhez vezetnének, ha azonnal aktiválnánk rájuk egy funkciót.

A legalább három egyezés alkalmazása egy kompromisszum a megbízhatóság és a reakcióidő között. Ha kevesebb egyezést várnánk el (pl. 1 vagy 2), akkor a rendszer túl érzékennyé válna, míg ha több egyezést (pl. 5 vagy több) írunk elő, akkor a felhasználó által várt gyors válaszidő szenvedne csorbát. A három ismétlés már kellően robusztus eredményt nyújt, miközben a felhasználói élmény sem sérül.

5.1.5. Webhook-alapú értesítések

A felismerés után a `notify_subscribers()` függvény segítségével történik a feliratkozott kliensek értesítése, ahogy a lenti kódrészletben is látszik. Ez lehetővé teszi, hogy például egy frontend alkalmazás vagy más szolgáltatás valós időben reagáljon a kézmozdulatokra.

```
1 async def notify_subscribers(gesture):
2     for url in list(subscribers):
3         try:
4             requests.post(url, json={"gesture": gesture})
5         except Exception as e:
6             print(f"Failed to notify {url}: {e}")
```

5.2. Oktatást segítő prezentációs alkalmazás

A frontend fejlesztése **React.js** alapokon valósult meg. A React egy népszerű, komponensalapú JavaScript könyvtár, amely lehetővé teszi a nagy teljesítményű, újrafelhasználható és dinamikus felhasználói felületek létrehozását. A választás fő oka, hogy a React ökoszisztémája széleskörű közösségi támogatással rendelkezik, és ideális egy modern, reszponzív oktatási webalkalmazás gyors fejlesztéséhez.

A React előnye, hogy egyirányú adatfolyamot és hatékony DOM-kezelést (Virtual DOM) biztosít, amely optimalizálja a felület újrenderelését, különösen akkor, ha gyakori vizuális frissítésekre van szükség – ilyen például az élő képszegmentálás vagy az interaktív animációk. Az alkalmazás több kulcsfontosságú funkciót is tartalmaz: ilyen a **többsz nyelvűség**, amelyet például az **i18next** könyvtár segítségével valósítottam meg, illetve a **MediaPipe** integráció, amely lehetővé teszi a valós idejű képszegmentálást közvetlenül a böngészőben.

A MediaPipe futtatása **WebGL 2**-t támogató böngészőkben történik, amely hardveres gyorsítást biztosít a képfeldolgozási műveletekhez. Ez lehetővé teszi, hogy a gépi látáson alapuló funkciók (például a háttér eltávolítása) zökkenőmentesen és erőforrás-hatékonyan működjenek kliensoldalon, anélkül, hogy a szerveret terhelnék.

A vizuálisan letisztult, felhasználóbarát kezelőfelület kialakításához a **Material UI** (MUI) könyvtárat használtam, amely a Google által megalkotott Material Design irányelveken alapul. A Material UI segítségével egységes, esztétikus és modern megjelenésű komponenseket tudtam létrehozni, amelyek illeszkednek az oktatási alkalmazás igényeihez. A könyvtár számos kész felhasználói interfész elemet biztosít (például gombok, kártyák, űrlapok), amelyek nemcsak jól néznek ki, hanem könnyen testreszabhatók és reszponzívak is.

5.2.1. Feliratkozás az API gesztusfelismerésére

A felhasználó a „Gesztus felismerés” gombra kattinva feliratkozhat az API-ra. Az API /subscribe_webhook endpointjára elküld egy URL-t HTTP POST kérésként, ami fogadni fogja a felismert gesztusról küldött push értesítést. A lenti kódrészletben bemutatom a feliratkozást.

```
1 const subscribeToWebhook = async () => {
2   try {
3     const response = await fetch(`${SERVER_URL}/
4     subscribe_webhook?url=${WEBHOOK_URL}`, {
5       method: "POST",
6       headers: {
7         "Content-Type": "application/json",
8       },
9     });
10    startFrameStreaming();
11  } catch (error) {
12    setError(error.message);
13  }
14};
```

Ezután elkezdni küldeni a startFrameStreaming() függvénnyel a kamera képét jpg-ként a /process_frame endpointra. A lenti példában a SERVER_URL konstans tartalmazza a /process_frame endpointot.

```
1 const startFrameStreaming = () => {
2   ...
3   const formData = new FormData();
4   formData.append("frame", blob, "frame.jpg");
5   try {
6     const response = await fetch(`${SERVER_URL}/
7     process_frame`, {
8       method: "POST",
9       body: formData,
10    });
11  } catch (error) {
12    console.error("Error sending frame", error);
13  }
14  ...
15}
```

5.2.2. Értesítés fogadása az API-tól

Az API-tól egy külön Express szerver fogadja az értesítéseket, amely ezt követően továbbítja az információt a React alkalmazásnak. Ennek oka technológiai és architekturális szempontból is indokolt.

A React önmagában egy kliensoldali JavaScript könyvtár, amely a böngészőben fut, így nem képes közvetlenül fogadni HTTP POST kéréseket. A webhookok azonban tipikusan szerveroldali hívások: a gesztusfelismerő API szerver azonnal értesítést küld, amikor új gesztust ismer fel, mégpedig egy HTTP POST kérés formájában. Ezért szükséges egy szerveroldali komponens, amely képes ilyen POST kérések fogadására és megfelelő feldolgozására.

Ebben az architektúrában az Express.js egy Node.js-alapú szerverkeretként működik, amely gyorsan konfigurálható és hatékonyan kezeli az aszinkron eseményeket. Az Express szerver fogadja az API által küldött webhookokat, dekódolja a kapott adatokat, majd websocketen keresztül vagy REST végponton továbbítja az információt a React frontend felé. Így a React alkalmazás valós időben tud reagálni a felismerésekre.

A FastAPI azért küld webhookot, mert az API célja nem csupán a válaszküldés a kliens által indított lekérdezésekre, hanem eseményvezérelt értesítések küldése is, például egy gesztus felismerése esetén. A webhook mechanizmus lehetővé teszi, hogy az API aktívan értesítse az előre regisztrált klienseket – jelen esetben az Express szervert – az esemény bekövetkeztekor, ezzel biztosítva a gyors, valós idejű kommunikációt.

Ez az architekturális megközelítés több előnnyel is jár:

- **Modularitás:** az egyes komponensek – gesztusfelismerő API, értesítéseket fogadó szerver, felhasználói felület – jól elkülönülnek egymástól.
- **Biztonság és skálázhatóság:** a bejövő webhookokat szerveroldalon validálni lehet, naplózni, vagy akár hitelesíteni.
- **Kiterjeszthetőség:** a szerver a későbbiekben bővíthető más szolgáltatások vagy több kliens kiszolgálására is, anélkül hogy a React oldalon változtatni kellene.

5.2.3. Kép szegmentálása

A képszegmentálás megvalósítása böngészőben különösen nagy technológiai kihívást jelent, mivel a valós idejű videófeldolgozáshoz nagy számítási kapacitásra van szükség. Ennek ellenére a modern grafikus webes API-k, mint például a **WebGL 2**, lehetővé teszik az ilyen jellegű műveletek böngészőoldali végrehajtását, GPU-gyorsítással. A szegmentálás során a cél a háttér és az előtér (jellemzően az emberi alak) szétválasztása, amely alapját képezi a további képfeldolgozási lépéseknek, például a háttér elmosásának, cseréjének vagy interaktív elemekhez történő illesztésének.

A projektben a **MediaPipe SelfieSegmentation** könyvtárat használok, amely a Google által fejlesztett valós idejű képfeldolgozó keretrendszer része. Ez a modul

mélytanuláson alapuló modellt alkalmaz az emberi alak elkülönítésére. Hatékonyan futtatható böngészőben kliensoldalon is.

A következő kódrészlet bemutatja a szegmentálás gyakorlati megvalósítását.

```
1 import { useRef, useEffect } from "react";
2 import Webcam from "react-webcam";
3 import { SelfieSegmentation } from "@mediapipe/
  selfie_segmentation";
4 import * as cam from "@mediapipe/camera_utils";
5
6 const webcamRef = useRef(null);
7 const canvasRef = useRef(null);
8
9 useEffect(() => {
10   const selfieSegmentation = new SelfieSegmentation({
11     locateFile: (file) => {
12       return 'https://cdn.jsdelivr.net/npm/@mediapipe/
13         selfie_segmentation/${file}';
14     },
15   });
16
17   selfieSegmentation.setOptions({
18     modelSelection: 1, // 0 - landscape, 1 - selfie
19     optimalizált
20   });
21
22   selfieSegmentation.onResults(onResults);
23
24   if (webcamRef.current !== null) {
25     const camera = new cam.Camera(webcamRef.current.video, {
26       onFrame: async () => {
27         await selfieSegmentation.send({ image: webcamRef.
28           current.video });
29       },
30       width: 1280,
31       height: 720,
32     });
33     camera.start();
34   }
35 }, []);
```

Ebben a megoldásban a `react-webcam` segítségével valós idejű videofelvételt készítünk, amit a `MediaPipe SelfieSegmentation` modul dolgoz fel.

A `modelSelection:1` opcióval a modellt kifejezetten emberi szelfik feldolgozására

optimalizáljuk, ami pontosabb szegmentálási eredményeket nyújt. A `camera_utils` modul a videófolyam képkockáit valós időben továbbítja a feldolgozó algoritmusnak.

A háttérszegmentálás eredménye egy maszkolt kép, amely alapján könnyen módosítható a háttér (pl. elhomályosítható, kicserélhető), vagy interaktív felhasználói élményt nyújtó vizuális elemek helyezhetők el kizárólag a felhasználó köré.

A technológia kiválasztása – azaz a MediaPipe és WebGL használata – annak köszönhető, hogy ezek lehetővé teszik a gépi tanulási modellek böngészőben történő futtatását, külön telepítés nélkül, natív alkalmazás szintű teljesítménnyel. Ez a megközelítés kompatibilis a legtöbb modern eszközzel, nem igényel háttérszervert a képfeldolgozáshoz, és jelentősen csökkenti a válaszidőt, így ideális interaktív oktatási alkalmazások számára.

5.2.4. Prezentációs fájlok kezelése

Az alkalmazás lehetővé teszi különböző prezentációs fájltypusok – például PDF dokumentumok, PowerPoint bemutatók (PPT, PPTX), valamint képfájlok – betöltését és megjelenítését. A felhasználó a „Fájl feltöltése” gomb segítségével kiválaszthatja a kívánt fájlt, amelyet az alkalmazás felismer, és a fájltypusnak megfelelő módon dolgoz fel és jelenít meg.

A prezentációs fájlok kezeléséhez több különböző, korszerű JavaScript könyvtárat használunk:

- **pdfjs-dist** – A Mozilla által fejlesztett PDF.js könyvtár, amely lehetővé teszi PDF dokumentumok beolvasását és renderelését HTML5 canvas elemekre.
- **PptxGenJS** – Egy PowerPoint (PPTX) fájlok létrehozására és manipulálására alkalmas JavaScript könyvtár. A projektben elsősorban a meglévő bemutatók tartalmának beolvasására használom.
- **html2canvas** – Ezzel a könyvtárral HTML elemekről készíthetünk képernyőképeket, amelyet opcionálisan exportálhatunk képként vagy prezentációs diába.
- **GlobalWorkerOptions** – A **pdfjs-dist** részeként a PDF.js worker szálának konfigurálásához szükséges, amely lehetővé teszi a nagyobb teljesítményű PDF renderelést.

A következő kódrészlet bemutatja, hogyan azonosítja és dolgozza fel az alkalmazás például a PDF fájltypusokat.

```

1 const imageHandler = async (e) => {
2   GlobalWorkerOptions.workerSrc = 'https://cdnjs.
   cloudflare.com/ajax/libs/pdf.js/2.10.377/pdf.worker.min.
   js';
3   const file = e.target.files[0];
4   const fileType = file.type;
5   setFileType(fileType);
6   if (fileType === "application/pdf") {
7     const pdfUrl = URL.createObjectURL(file);
8     setPdfUrl(pdfUrl);
9     const pdf = await pdfjsLib.getDocument(pdfUrl).
   promise;
10    const totalPages = pdf.numPages;
11    setTotalPdfPages(totalPages);
12    onPdfPageChange(pdfPageNum);
13  }
14 }

```

Az alkalmazásom három fájlípust különböztet meg, amiket ezekkel a módszerekkel dolgoz fel:

- **PDF fájlok:** A fájl egy URL-re kerül konvertálásra a `URL.createObjectURL` függvénnyel, majd a `PDF.js` segítségével betöltjük és lekérjük az oldalszámot. Ez lehetővé teszi az oldalak közötti navigációt.
- **PowerPoint fájlok:** A PowerPoint bemutatókat a `FileReader` segítségével olvassuk be `ArrayBuffer` formátumban, majd a `PptxGenJS` könyvtár segítségével feldolgozzuk a diákat.
- **Képfájlok (pl. JPG, PNG):** Ezeket a fájlokat szintén a `FileReader` segítségével töltjük be, és `base64` formátumban állítjuk elő őket a megjelenítéshez.

A technológiai választások azért estek erre, mert ezek a könyvtárak széles körben használt, jól dokumentált és aktívan fejlesztett eszközök, amelyek támogatják a valós idejű fájl megjelenítést böngészőben. Emellett lehetővé teszik a platformfüggetlen működést, így az alkalmazás külön telepítés nélkül is képes kezelni a különféle prezentációs formátumokat.

5.2.5. Szegmentált kép megjelenítése a vásznon

A felhasználó valós idejű videóképe a `MediaPipe` által szegmentálásra kerül, amely során a háttér és az előtér (általában a személy) elkülönül egymástól. A `MediaPipe SelfieSegmentation` könyvtára ehhez egy *segmentation mask*-ot generál, amely binárisan jelöli, hogy a képen mely pixelek tartoznak az előtérhez.

A szegmentált kép megjelenítéséhez az alkalmazás egy HTML5 canvas elemet használ, amelyre a feldolgozott képet rajzoljuk ki. A megjelenítés során több lépésben kombináljuk a videóképet, a maszkolt réteget, valamint egy háttérképet. A vászon renderelését a következő kódrészlet végzi:

```
1 const onResults = async (results) => {
2   const img = document.getElementById('vbackground')
3   const videoWidth = webcamRef.current.video.videoWidth;
4   const videoHeight = webcamRef.current.video.videoHeight;
5
6   canvasRef.current.width = videoWidth;
7   canvasRef.current.height = videoHeight;
8
9   const canvasElement = canvasRef.current;
10  const canvasCtx = canvasElement.getContext("2d");
11
12  canvasCtx.save();
13  canvasCtx.clearRect(0, 0, canvasElement.width,
14    canvasElement.height);
15
16  canvasCtx.drawImage(results.image, 0, 0, canvasElement.
17    width, canvasElement.height);
18  canvasCtx.globalCompositeOperation = 'destination-atop';
19  canvasCtx.drawImage(results.segmentationMask, 0, 0,
20    canvasElement.width, canvasElement.height);
21
22  canvasCtx.globalCompositeOperation = 'destination-over';
23  canvasCtx.drawImage(img, 0, 0, canvasElement.width,
24    canvasElement.height);
25  canvasCtx.restore();
26 }
```

A megoldás alapjául szolgáló implementáció megtalálható itt [27].

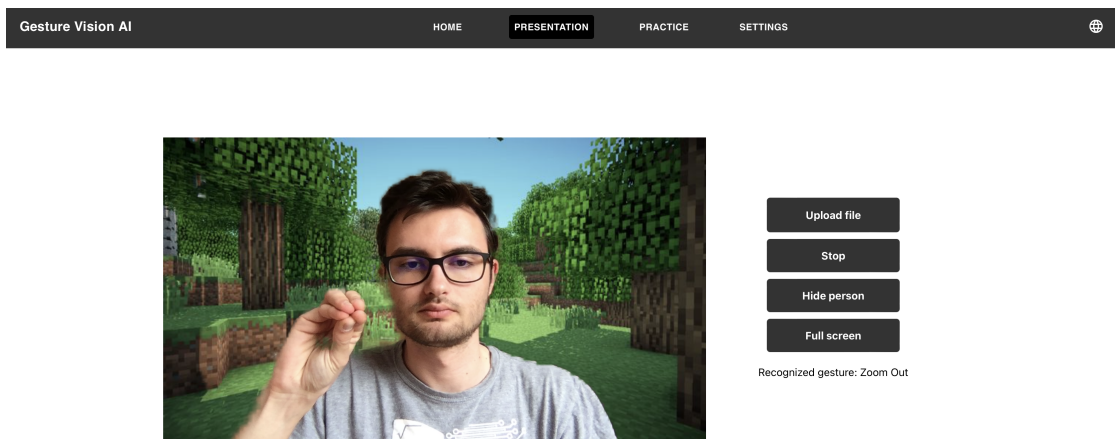
A fenti kódrészlet meggyarázata

- A videoWidth és videoHeight értékeket a webkamera stream méreteiből olvassuk ki, és ehhez igazítjuk a vászon méretét is.
- Az első drawImage hívás a teljes színes képet rajzolja a vászonra, méretezve, középre igazítva.
- A globalCompositeOperation = 'destination-atop' beállítás gondoskodik arról, hogy csak azok a képpontok maradjanak láthatóak, amelyek az előtérhez tartoznak (a maszk alapján).

- Ezután a `globalCompositeOperation = 'destination-over'` opcióval egy háttérkép kerül a szegmentált személy mögé – ez lehet statikus kép, másik videó, vagy bármilyen grafikai elem.

A kép megjelenítése WebGL 2 támogatással történik, mivel a HTML5 canvas 2D API-ja a modern böngészőkben már GPU-gyorsítással működik, így képes folyamatos, valós idejű renderelésre. Ez a megközelítés lehetővé teszi, hogy extra bővítmények vagy külső alkalmazások nélkül, csupán a böngészőben végezzük el a szegmentálást és annak megjelenítését. A `MediaPipe SelfieSegmentation` könyvtárat azért választottam, mert:

- Valós idejű működést biztosít még alacsonyabb teljesítményű eszközökön is.
- Kifejezetten emberi sziluettek szegmentálására lett optimalizálva.
- Könnyen integrálható React alapú projektekbe, különösen a `@mediapipe/camera_utils` könyvtárral kombinálva.



5.1. ábra. Az alkalmazás Presentation oldala

Ez a megközelítés lehetővé teszi, hogy a felhasználó saját magát „beilleszthesse” egy virtuális háttérbe, amit a prezentáció során a rendszer a videóképpel együtt jelenít meg.

5.2.6. Gesztus alapú interakciók

A gesztusfelismeréshez használt MediaPipe és a gesztusokhoz rendelt műveletek lehetővé teszik, hogy a felhasználó a bemutató közben kéz gesztusokkal navigáljon. Az alábbiakban egy egyszerű példát találunk, amely a „Swipe Right” gesztust használja a PDF oldalak előre navigálásához.

```
1 if (gesture) {  
2     const mappedGesture = storedSettings[gesture] || gesture  
3     ;  
4     if (mappedGesture === "Swipe Right") {  
5         if (fileType === "application/pdf" && pdfPageNum <  
6             totalPdfPages) {  
7             setPdfPageNum((prev) => prev + 1);  
8         }  
9     }  
10 }
```

Ez a kódrészlet egy egyszerű gesztusalapú interakciót valósít meg, amely lehetővé teszi a PDF-oldalak közötti navigációt a felhasználó kézmozdulatai alapján. A `gesture` változó egy, az API által detektált kézmozdulatot tartalmaz, például „Swipe Right” (jobbra húzás). A rendszer először leképezi ezt a gesztust az esetleges felhasználói beállítások (`storedSettings`) alapján – ez lehetőséget biztosít arra, hogy a felhasználó saját mozdulatokat rendeljen adott műveletekhez.

Amennyiben a leképzett gesztus „Swipe Right”, és az aktuálisan megnyitott fájl típusa PDF, valamint nem értünk még a dokumentum végére, akkor a `setPdfPageNum` függvény segítségével a következő oldalra lapozunk.

A `setPdfPageNum((prev) => prev + 1)` konstrukció a React állapotkezelésében szokásos minta, amely mindig az aktuális oldalszámhoz képest növeli eggyel az értéket.

5.2.7. Laser pointer

A Laser Pointer funkció célja, hogy a felhasználó az ujjával mutathasson a képernyőn, mintha egy lézermutatót használna. Ez különösen hasznos lehet prezentációk során, amikor a beszélő vizuálisan szeretne rámutatni egy adott tartalomra anélkül, hogy közvetlenül megérintené az eszközt vagy használná az egeret. A rendszer működése két komponensből áll:

1. A **MediaPipe Hands** segítségével az API folyamatosan elemzi a React alkalmazástól kapott képkockákat, és felismeri az `INDEX_FINGER_TIP` (mutatóujj hegye) koordinátáit.

2. Ezeket a koordinátákat az API egy szöveges push értesítésben visszaküldi a kliensnek, ahol a React alkalmazás egy piros pontot rajzol az adott pontra, imitálva ezzel egy lézermutatót.

Az API kiküldi push értesítésként a detektált mutatóujj x és y pixel koordinátáit a lenti kódrészletben látható módon.

```

1     if landmarks:
2         for hand_landmarks, handedness in zip(landmarks,
handedness_list):
3             index_tip = landmarks[self.mp_hands.HandLandmark
.INDEX_FINGER_TIP]
4             x, y = index_tip.x, index_tip.y
5             x_pixel = int(x * frame.shape[1])
6             y_pixel = int(y * frame.shape[0])
7             return f"{x_pixel},{y_pixel}"

```

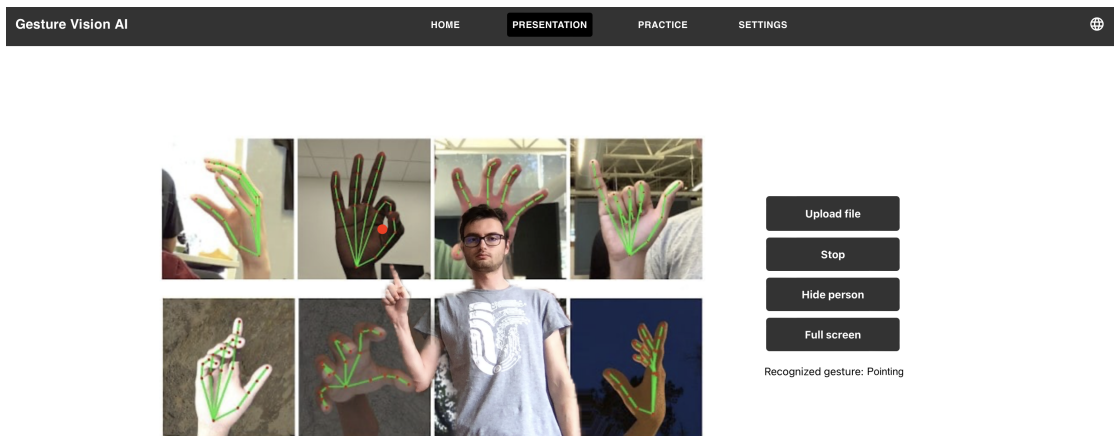
A React oldalon a lézerpointer kirajzolása pedig a következő módon történik:

```

1 const [coords, setCoords] = useState(null);
2 useEffect(() => {
3     const canvas = canvasRef.current;
4     if (!canvas || !coords) return;
5
6     const ctx = canvas.getContext("2d");
7     ctx.clearRect(0, 0, canvas.width, canvas.height);
8     ctx.beginPath();
9     ctx.arc(coords.x, coords.y, 8, 0, 2 * Math.PI);
10    ctx.fillStyle = "red";
11    ctx.fill();
12 }, [coords]);

```

Ez a megoldás teljesen érintésmentes és rendkívül látványos. A MediaPipe gyors és pontos kézmozdulat-felismerésének köszönhetően a felhasználó ujjának mozgása valós időben követhető a képernyőn, ezzel kiváltva a klasszikus lézermutató eszközöket. Az 5.2. ábrán látható a működése. A felhasználó a mutató ujját használva egy piros pontot irányít a képernyőn.



5.2. ábra. Laser pointing, ujj használata mutatóra, egér kurzor nélkül

5.2.8. i18n - Többnyelvűség támogatása

A többnyelvűség kezelése a React alkalmazásban a **i18n** (internationalization) könyvtár segítségével történik. Az alkalmazás három nyelvet támogat: angolt, magyart és román nyelvet. A nyelvi fájlok (pl. `hu/translation.json`) tartalmazzák az összes szükséges fordítást az adott nyelvhez. Az i18n rendszer lehetővé teszi a felhasználó számára, hogy az alkalmazásban válasszon a három támogatott nyelv közül.

A nyelvek közötti váltás az alábbi módon valósul meg.

```

1 import { useTranslation } from 'react-i18next';
2 const { i18n } = useTranslation();
3
4 const handleLanguageChange = (newLanguage) => {
5   i18n.changeLanguage(newLanguage);
6   setOpen(false);
7 };

```

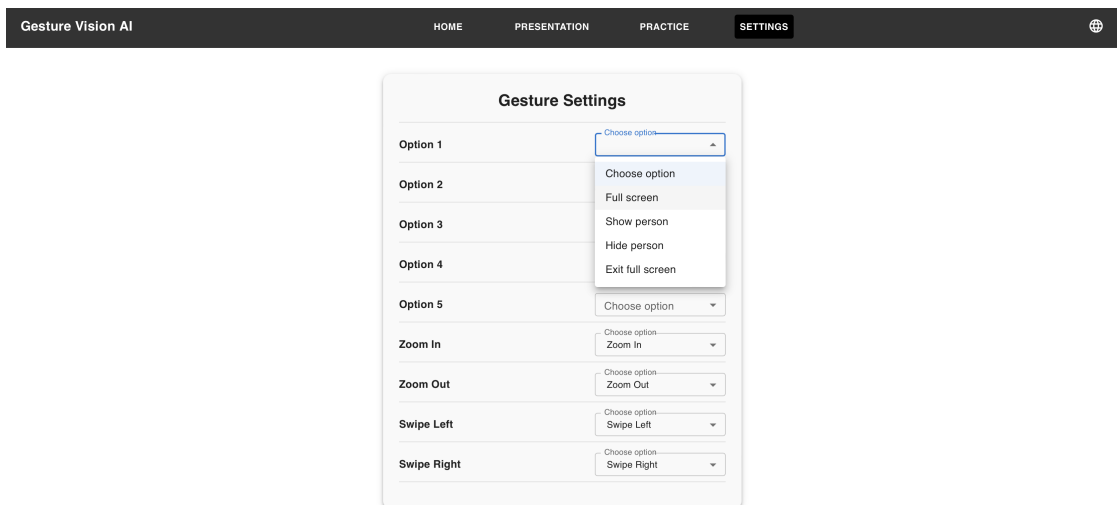
Ez a komponens egyszerűsített módon teszi lehetővé a felhasználók számára a nyelv váltását, miközben minden egyes oldalon a megfelelő fordítások kerülnek alkalmazásra.

Példaként bemutatnám, hogy így néz ki a `hu/translation.json` fájl, ami a magyar fordításért felel:

```
1 {  
2   "menuBar": {  
3     "home": "Főoldal",  
4     "presentation": "Prezentáció",  
5     "practice": "Gyakorlás",  
6     "settings": "Beállítások"  
7   },  
8   ...  
9 }
```

5.2.9. Funkciók beállítása a felismert gesztusra

Az alkalmazás lehetőséget biztosít arra, hogy a felhasználó saját igényei szerint rendelje hozzá az egyes gesztusokat különféle funkciókhoz. Ezt a **Settings** oldalon keresztül lehet megtenni, ahol egy egyszerű legördülő menüből választható ki, hogy például a **Swipe Right** gesztus prezentáció lapozást, nagyítást vagy akár teljes képernyős nézetet váltson ki. Így a prezentáció vezérlése teljes mértékben testre szabható, és igazítható a felhasználó szokásaihoz.



5.3. ábra. Az alkalmazás Beállítások oldal

Az 5.3. ábrán látható a beállítások oldal. A bal oszlopban a gesztusk vannak megjelenítve. Minden gesztusnak ki tudunk választani egy funkciót egy legördülő listából, hogy mit csináljon. Például az „Option 1” gesztusnak be tudjuk állítani, hogy tegye teljes képernyős nézetbe a prezentációt. Látható az is, hogy négy

gesztusnak már alaphól be van állítva funkció, ez alapbeállítás, de a felhasználó módosíthatja ezt.

A beállításokat az alkalmazás `sessionStorage`-ben tárolja, így azok az oldal bezárásáig megmaradnak, és újratöltés után is automatikusan érvénybe lépnek. A választás oka, hogy ezek a gesztus-beállítások tipikusan munkamenet-specifikus jellegűek: azaz nem cél, hogy tartósan el legyenek mentve minden későbbi látogatáshoz, hanem inkább az adott használati alkalomhoz illeszkedjenek. A `sessionStorage` használata lehetővé teszi az adatok automatikus törlését, amikor a felhasználó bezárja a böngészőablakot, ezáltal biztosítva a kontextusfüggő működést és a rendszer erőforrásainak hatékonyabb kezelését.

Az alábbi kódrészlet szemlélteti, hogyan történik a mentett beállítások betöltése és inicializálása a React komponens indulásakor.

```
1 useEffect(() => {  
2     const storedSettings = sessionStorage.getItem("  
3     gestureSettings");  
4     if (storedSettings) {  
5         setSettings(JSON.parse(storedSettings));  
6     } else {  
7         setSettings(defaultSettings);  
8         sessionStorage.setItem("gestureSettings", JSON.  
9         stringify(defaultSettings));  
10    }  
11 }, []);
```

Ez a React `useEffect` hook segítségével ellenőrzi, hogy található-e elmentett gesztus-beállítás a `sessionStorage`-ben. Amennyiben van ilyen, azt betölti és alkalmazza, ellenkező esetben az alapértelmezett értékeket használja, és el is menti azokat a munkamenet idejére. A **Settings** oldal kialakítása lehetővé teszi, hogy minden elérhető gesztushoz egyedi funkció legyen rendelve, ami személyre szabható és bővíthető felhasználói élményt biztosít.

5.2.10. Gyakorló felület

Az alkalmazás részeként elérhető egy Gyakorló oldal (**Practice Page**), amely lehetőséget biztosít a gesztusok begyakorlására.

A gyakorló oldalon minden gesztus egy-egy különálló **Card** komponensként jelenik meg. A felhasználó rákattinthat az adott gesztust jelző kártyára, amelyre a rendszer kameraképet jelenít meg, és valós időben figyeli a kézmozdulatokat az API segítségével. Ebben az esetben is továbbítjuk a képet az API-nak, ami visszaadja éresítésként a felismert gesztust.

Amennyiben a felhasználó a kiválasztott gesztust helyesen mutatja be, az alkalmazás visszajelzést ad — egy zöld színű ikon és egy felirat segítségével —, jelezve, hogy a mozdulat megfelel a gesztus mintájának. Ezzel a visszajelzéssel a felhasználó azonnal láthatja, hogy jól hajtotta-e végre a kívánt interakciót.

5.3. Gesztusvezérelt okos lámpa

5.3.1. Használt hardverelemek és bekötések

A rendszer fő vezérlőegysége egy **ESP8266 NodeMCU mikrokontroller**, amely Wi-Fi kapcsolaton keresztül képes API-ból érkező gesztusadatok fogadására. A kapott gesztusok alapján a mikrokontroller egy relén keresztül vezérli a 230V-os hálózatról táplált lámpát, valamint az aktuális gesztust megjeleníti egy LCD kijelzőn.

- **ESP8266 NodeMCU**: a központi vezérlő, amely a Wi-Fi kapcsolatot, gesztusértékelést és relévezérlést valósítja meg.
 - D1 (GPIO5) – relé vezérlőjele. A mikrokontroller ezen a lábon keresztül kapcsolja a relét be vagy ki.
 - D2 (GPIO4) – I2C SDA (adatvonal), az LCD kijelzővel való kommunikációhoz.
 - D3 (GPIO0) – I2C SCL (órjelvonal), szintén az LCD kommunikációhoz.
- **Hi-Link HLK-PM03 AC-DC átalakító**: a 230V váltakozó feszültséget alakítja át 3.3V egyenfeszültséggé, amely közvetlenül táplálja az ESP8266 mikrokontrollert. Az átalakító és a relé **párhuzamosan vannak kötve**, így mindkettő ugyanabból az AC forrásból kap tápot.
- **Tongling 5V relémodul**: elektromos kapcsolóként működik, amely az ESP8266 vezérlőjére nyitja vagy zárja a lámpa fázisvezetékét.
 - A 230V AC **fázisvezetéke** megy be a relé **C (Common)** lábára.
 - A **N0 (Normally Open)** kimeneten keresztül jut a fázis a lámpába, így az csak akkor kap feszültséget, ha a relé zárt állapotban van.
 - Az **NC (Normally Closed)** láb nem kerül bekötésre.
- **LCD kijelző (I2C interfésszel)**: az aktuálisan felismert gesztus megjelenítésére szolgál. Az I2C interfész lehetővé teszi, hogy a kijelzőt csak két vezetékkel (SDA és SCL) csatlakoztassuk az ESP8266-hoz, így GPIO lábakat takarítunk meg.

- **230V-os AC lámpa:** a relén keresztül vezérelt terhelés. A lámpán található P és N jelölések a fázis és nulla bekötési pontokat jelölik. A relé a **fázist szakítja meg**, mivel ez biztonságosabb, mint a nulla oldalon való kapcsolás (a fogyasztó így feszültségmentes lesz kikapcsolt állapotban).

5.3.2. Bekötési logika és biztonsági megfontolások

A Hi-Link tápegység és a relémodul **párhuzamos bekötése** az AC hálózatra lehetővé teszi, hogy az ESP és a relé egyszerre kapjanak tápellátást. A relé által kapcsolt fázisvezeték biztonságosabb működést biztosít, hiszen így kikapcsolt állapotban a lámpa nem marad feszültség alatt.

Az LCD kijelző I2C interfésze jelentősen leegyszerűsíti a kommunikációt a mikrokontroller és a kijelző között, hiszen csak két adatvonalat (SDA, SCL) használ, így több GPIO marad más perifériák számára.

A D1 láb kiválasztása a relévezérléshez az ESP8266 alapértelmezett GPIO kiosztásához igazodik, amely ezen a lábon megbízhatóan tud digitális jelet szolgáltatni. A D2 és D3 lábak szintén gyakori I2C kiosztások NodeMCU esetén, így azok használata kompatibilis legtöbb könyvtárral és példaprogrammal.

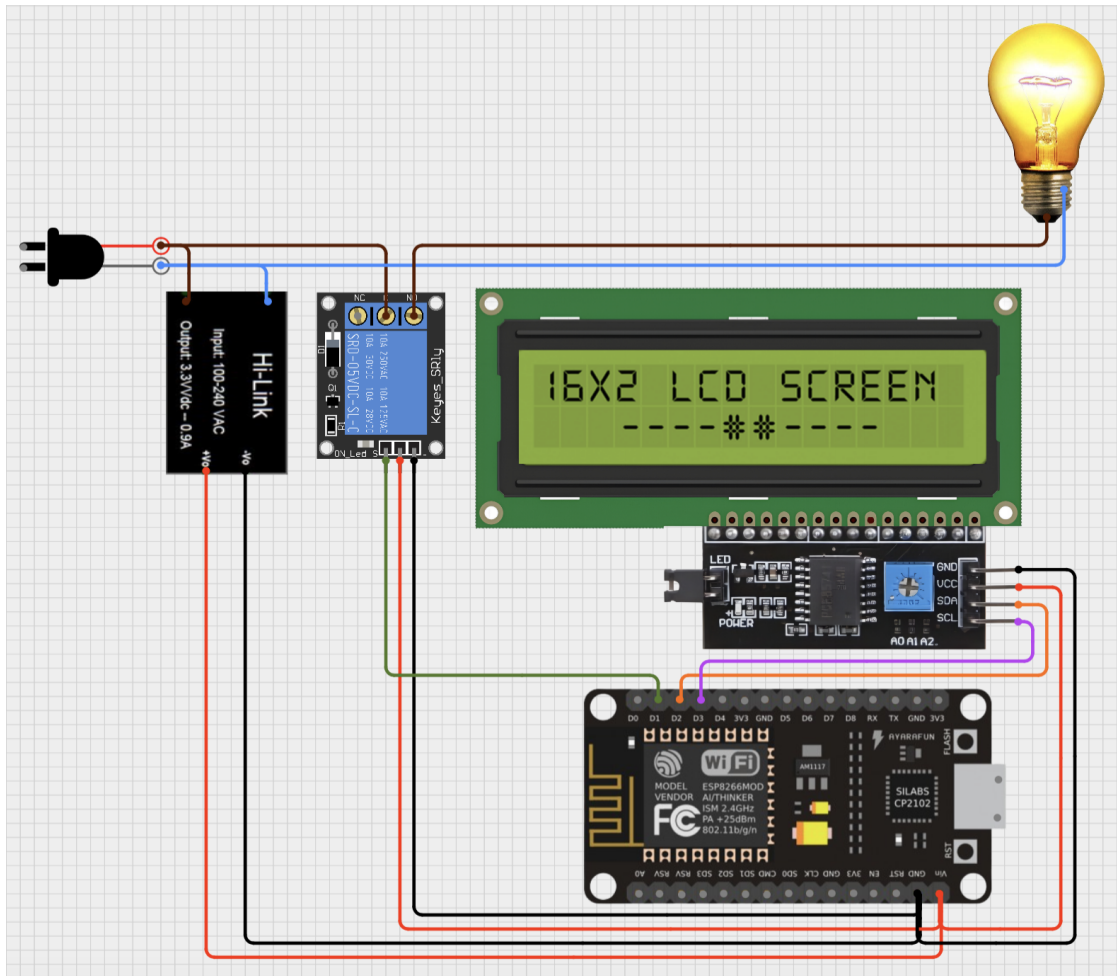
5.3.3. ESP8266, NodeMCU és GPIO lábak szerepe

A rendszer lelke az **ESP8266** mikrokontroller, amely egy alacsony fogyasztású, nagy integráltságú Wi-Fi chip beépített processzorral. Alapvetően az ESP8266 egy önállóan is működő, programozható egység, amely képes TCP/IP protokollok használatára és mikrovezérlői feladatok ellátására.

A projektben az ESP8266 **NodeMCU** fejlesztőlap formájában lett használva. A NodeMCU egy olyan nyílt forráskódú fejlesztőpanel, amely az ESP8266 chipre épül, és USB-ről táplálható, valamint előre kihozott **GPIO** (General Purpose Input/Output – általános célú bemenet/kimenet) lábakat biztosít.

GPIO lábak: Ezek a mikrokontroller olyan csatlakozási pontjai, amelyeken keresztül a külső eszközökkel (pl. relé, szenzorok, kijelzők) történő kommunikáció megvalósítható. A GPIO lábakat szoftverből programozva be- vagy kimeneti módban lehet használni, és digitális jeleket (pl. 0V vagy 3.3V) küldhetnek vagy fogadhatnak.

- **D1 (GPIO5)** – Digitális kimenet, amellyel a relét vezérli (bekapcsolás/kikapcsolás).
- **D2 (GPIO4)** és **D3 (GPIO0)** – I2C kommunikációs vonalak (SDA, SCL), amelyek az LCD kijelzővel történő adatcserét végzik.



5.4. ábra. Az okos lámpa kapcsolási rajza

5.3.4. A Wi-Fi alapú kommunikáció előnye

Az ESP8266 legnagyobb előnye, hogy beépített Wi-Fi modullal rendelkezik, így közvetlenül képes csatlakozni egy helyi hálózathoz vagy akár az internetre. Ebben a rendszerben ez teszi lehetővé, hogy a mikrokontroller a gesztusfelismerő API-val kommunikáljon.

A rendszer úgy működik, hogy a gesztusfelismerés egy kamera által történik, amely továbbítja az adatokat az API-nak, majd az API feldolgozza és egy push értesítést küld az ESP8266-nak. Ez a push üzenet tartalmazza a felismert gesztust, például: „option 1” -> kapcsolja be a lámpát.

- A Wi-Fi-n keresztüli kommunikáció révén az ESP8266 valós időben tud reagálni az eseményekre, nincs szükség fizikai gombokra vagy kapcsolókra.

- A megoldás távolról is vezérelhető.
- A vezérlés teljesen vezetékmentes, így ideális okosotthon rendszerekhez.

5.3.5. Miért ezeket az eszközöket használtam?

- **ESP8266:** Nagyon kedvező ár-érték arányú, Wi-Fi képes mikrokontroller, rengeteg könyvtár és közösségi támogatás áll rendelkezésre hozzá. Egyszerűen programozható az Arduino IDE segítségével, ahol az Arduino C/C++ nyelvet használhatjuk, és rengeteg kész könyvtár támogatja az ESP8266-ot, relét és LCD kijelzőt.
- **NodeMCU fejlesztőlap:** Könnyen csatlakoztatható USB-n keresztül, rendelkezik stabil 3.3V szabályzóval, és előre kihozott, számozott GPIO lábakkal, így ideális prototípusfejlesztéshez.
- **I2C - LCD kijelző:** A megszokott 16 vezetékes kijelzőkkel szemben ez csak 2 adatvezeték igényel (SDA és SCL), így kevesebb lábat foglal el az ESP-n. Ideális, ha több perifériát szeretnénk egyszerre használni.
- **Hi-Link HLK-PM03:** Megbízható és kis méretű AC-DC átalakító modul, amely közvetlenül az elektromos hálózathoz állít elő stabil 3.3V feszültséget az ESP számára.
- **Tongling 5V relémodul:** Egyszerű és jól dokumentált relé, amely közvetlenül vezérelhető az ESP 3.3V-os logikájáról, és lehetővé teszi 230V-os váltóáramú fogyasztók kapcsolását.

5.3.6. Kódrészlet

Ebben az alfejezetben bemutatok néhány C++-ban írt kódrészletet, ami az ESP8266-on fut.

A mikrokontroller addig vár, amíg sikeresen kapcsolódik az adott WiFi hálózathoz.

```
1 WiFi.begin(ssid, password);  
2 while (WiFi.status() != WL_CONNECTED) {  
3     delay(500);  
4 }
```

Az API JSON-formátumban küldi a gesztust egy HTTP POST kérdésben az ESP8266-on futó szerver url-jére. Az ESP8266 ezt fogadja, kinyeri az adatot, és továbbítja a megjelenítő és vezérlő logikának.

```
1 StaticJsonDocument<200> doc;  
2 DeserializationError error = deserializeJson(doc, body);  
3 String gesture = doc["gesture"];  
4 Serial.println("Received gesture: " + gesture);
```

Az aktuális gesztust kiírjuk az LCD-re, és ha az „option 1”, akkor bekapcsoljuk a relét. Minden más esetben kikapcsoljuk.

```
1 lcd.setCursor(0, 0);  
2 lcd.print("Gesture:");  
3 lcd.setCursor(0, 1);  
4 lcd.print(gesture);  
5 if (gesture == "option 1") digitalWrite(RELAY_PIN, HIGH); //  
    bekapcsol  
6 else digitalWrite(RELAY_PIN, LOW); // kikapcsol
```

5.4. Push értesítés küldése mobiltelefonra

A mobilalkalmazás Swift nyelven készült, mivel iOS és macOS készülékek álltak rendelkezésemre a fejlesztés során. Így natív élményt biztosít az Apple ökoszisztémán belül. A Swift alapú natív fejlesztés előnye, hogy az alkalmazás maximálisan kihasználja az iOS platform nyújtotta lehetőségeket – például gyorsabb működés, zökkenőmentes rendszerintegráció (értesítések, kamera, háttérműködés), valamint a felhasználói élmény is illeszkedik az Apple által meghatározott irányelvekhez. Emellett a SwiftUI és a Combine technológiák lehetővé teszik modern és könnyen karbantartható kód megvalósítását. A mobilalkalmazás képes fogadni az API által küldött push értesítéseket. Az alkalmazásban az `URLSession` és `NotificationCenter` segítségével regisztráljuk az endpointot, és fogadjuk a POST kéréseket.

5.4.1. Feliratkozás az API-ra

A mobilalkalmazás egy webhook URL-t küld a backend szervernek, amelyen keresztül később a gesztusértesítések érkeznek. A regisztráció HTTP POST kéréssel történik, ahol a kliens elküldi saját URL-jét az API-nak. Feliratkozás az API-ra így valósul meg, a lenti kódrészletben bemutatva.

```

1 func registerForWebhook() {
2     let url = URL(string: API_URL)!
3     var request = URLRequest(url: url)
4     request.httpMethod = "POST"
5     request.setValue("application/json", forHTTPHeaderField:
6         "Content-Type")
7     let body = ["url": MOBIL_DEVICE_URL]
8     request.httpBody = try? JSONSerialization.data(
9         withJSONObject: body)
10    URLSession.shared.dataTask(with: request) { data,
11        response, error in
12        if let error = error {
13            return
14        }
15        print("success")
16    }.resume()
17 }

```

URLSession az iOS beépített hálózati könyvtára, amely lehetővé teszi HTTP kérések indítását és aszinkron adatkommunikációt. **URLRequest** az adott HTTP kérés beállításait tartalmazza (metódus, fejlécek, body), míg a **JSONSerialization** a Swift objektumokat konvertálja JSON formátummá. Az így létrehozott POST kérés biztosítja, hogy a backend tudja, hová kell később a gesztusértesítést küldeni.

5.4.2. Értesítés fogadása az API-tól

Az alkalmazás egy saját backend szerveren keresztül fogadja a POST kéréseket, amelyeket az API küld, amikor egy gesztust felismer. Ez a függvény dekódolja a JSON objektumot, majd átadja az adatot az értesítési rendszernek.

```

1 func handleGesturePost(request: Request) throws ->
2     EventLoopFuture<HTTPStatus> {
3     let gesture = try request.content.decode(
4         GestureNotification.self)
5     sendLocalNotification(with: gesture.name)
6     return request.eventLoop.makeSucceededFuture(.ok)
7 }

```

Ez a kódrészlet a **Vapor** nevű Swift-alapú web framework használatán alapul. A **Request** típus a beérkező HTTP kérést reprezentálja. A `request.content.decode(...)` automatikusan dekódolja a JSON tartalmat

egy Swift struktúrába. Az `EventLoopFuture` a nem blokkoló, aszinkron feldolgozást teszi lehetővé, így az API válasza azonnal visszatéríthető a feldolgozás után.

5.4.3. Értesítés megjelenítése

A dekódolt gesztus nevét felhasználva a rendszer egy helyi értesítést generál a felhasználónak. Ez akkor is működik, ha az alkalmazás a háttérben fut, így a felhasználó azonnal értesülhet a felismerésről.

```
1 func sendLocalNotification(with gesture: String) {  
2     let content = UNMutableNotificationContent()  
3     content.title = "GESTURE VISION AI"  
4     content.body = "Felismert Gesztus: \(gesture)"  
5     content.sound = .default  
6  
7     let request = UNNotificationRequest(  
8         identifier: UUID().uuidString,  
9         content: content,  
10        trigger: nil  
11    )  
12  
13    UNUserNotificationCenter.current().add(request)  
14 }
```

Az iOS rendszerértesítéseit a **UserNotifications** keretrendszer biztosítja. A `UNMutableNotificationContent` objektum tartalmazza az értesítés szövegét, címét és hangját. A `UNNotificationRequest` egy egyedi azonosítóval ellátott értesítési eseményt hoz létre, amelyet az `UNUserNotificationCenter` kezel. Az értesítés megjelenítése teljes mértékben integrált az iOS rendszerrel (pl. zárolt képernyőn, notification centerben), így a felhasználó garantáltan észreveszi azt.



5.5. ábra. Értesítés iOS mobiltelefonon
Forrás: www.figma.com

6. fejezet

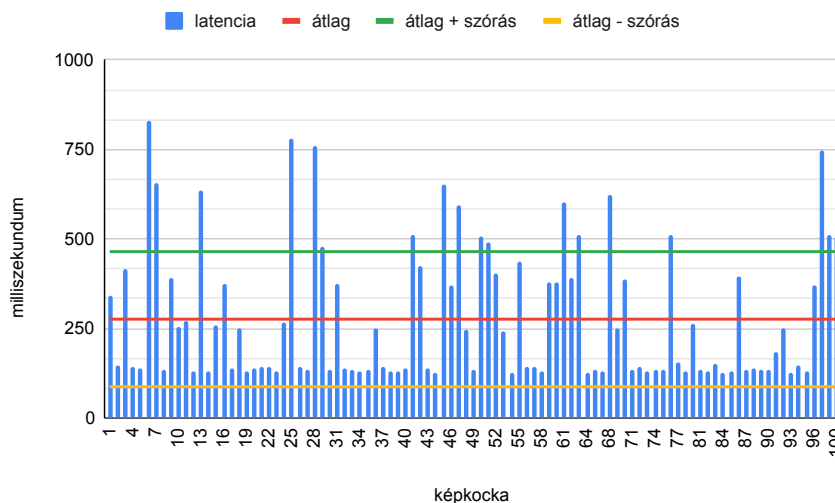
Mérések

6.1. Latencia(kliens - API között)

Ebben a mérésben a kliens és az API közötti válaszidők (latencia) elemzését végezzük el. Mind a kliens, mind az API localhost-on futnak, különböző portokon a saját számítógépemen. A mérések célja, hogy meghatározzuk, milyen gyorsan válaszol az API, és hogy az értékek mennyire stabilak az egyes frame-ek között. A mérés során 100 képkockát küldtem el az API-nak, és mértem azok válaszüdejét. Az oktatási segédprogramban a következő folyamat zajlott: elküldtem egy képkockát, lekértem a jelenlegi időt (startTime), majd amikor visszakaptam a felismert gesztust, ismét lekértem az időt (endTime). A két időpont közötti különbség adja meg a latenciát. A mérés a kódban látható módon történt: a `startFrameStreaming()` függvény indította el az idő mérését, amikor elkezdődött a képkockák küldése az API felé. A mérés a `useEffect()` hook-ban lett leállítva, amikor a válasz érkezett, és a gesztus visszaküldésre került a kliensnek a webhookon keresztül. A lenti kódrészletben látható a mérés indítása és leállítása.

```
1 const startFrameStreaming = () => {  
2   ...  
3   const startTime = performance.now();  
4   ...  
5 }  
6 useEffect(() => {  
7   ...  
8   const endTime = performance.now();  
9   const duration = endTime - startTime;  
10  ...  
11 }
```

A 6.1. ábrán látható, hogy 100 képkockából a legkisebb időtaram amíg az API válaszol 124 milliszekundum (ms), és a legtöbb 828 ms. Az átlagos válaszidő 276 ms. A szórás 188,7. Azt mutatja, hogy a válaszidők mennyire ingadoznak az átlag körül. Minél kisebb a szórás, annál inkább stabil az API válaszüzeje. Ezen a ponton fontos megjegyezni, hogy az elég magas szórás arra utal, hogy a válaszidők között jelentős ingadozások vannak, ami azt jelzi, hogy az API nem teljesít egyformán minden kérésnél.

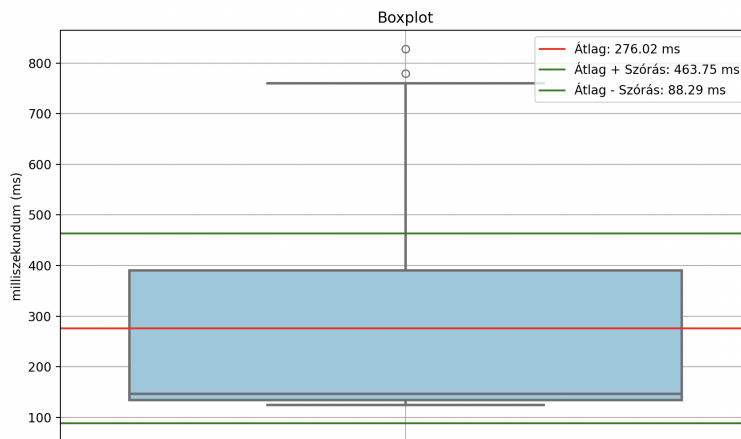


6.1. ábra. Latencia, átlag és szórás

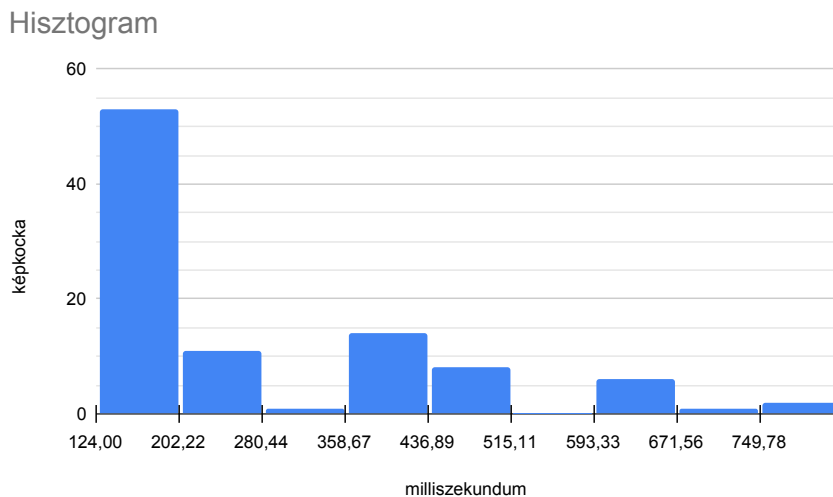
A 6.2. ábra az API válaszidők eloszlását mutatja. A doboz közepén lévő vonal az átlagos válaszidőt (276 ms) mutatja. A whisker vonalak a legkisebb és legnagyobb értékeket ábrázolják, amelyek még nem számítanak kiugró adatnak.

A szórás (standard deviation) megmutatja, mennyire szóródnak az értékek az átlag körül, azaz mennyire változó a mért időtartam (latencia) a különböző frame indexeknél. Ezzel állapítom, hogy mennyire stabil az API válaszüzeje.

Az 6.2. ábrán látható outlierok (kiugró értékek) azok az adatpontok, amelyek a whisker vonalakon túl helyezkednek el, és azokat az értékeket jelölik, amelyek jelentősen eltérnek a többi válaszidőtől.



6.2. ábra. Boxplot diagram



6.3. ábra. Hisztogram

A 6.3. ábra a hisztogramot mutatja be, amely az API válaszidők eloszlását szemlélteti. Az ábrán látható, hogy 100-ból a legtöbb(52 darab) képkocka 124 és 202,22 ms alatt került feldolgozásra és a válasz visszaküldésére a kliensnek. Ez azt jelenti, hogy a legtöbb képkocka 124 és 202,22 ms között lett kiszolgálva, vagyis ott van a tipikus válaszidő. A többi 48 képkockának tovább tartott a feldolgozása és a gesztus visszaküldése a kliensnek.

6.2. Gesztusfelismerés pontossága

A mérés során egy statikus („Zoom In” - nagyítás) és egy dinamikus („Swipe Right” - jobbra lapozás) gesztust ismert fel a rendszer. Mindkét esetben a kezem a kamerától 5-100 centiméter távolság között volt. Az API-ban létrehoztam egy **logger**-t amivel követtem a felismert gesztusokat.

```

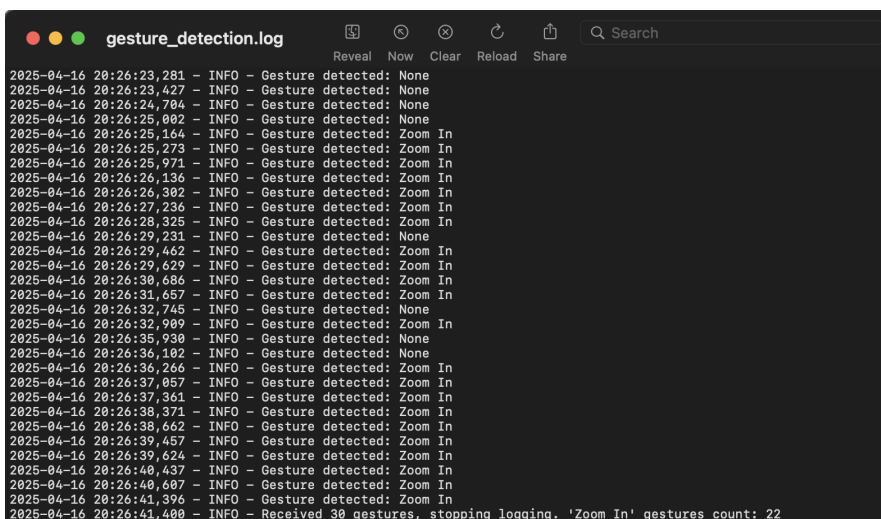
1 import logging
2
3 logging.basicConfig(
4     filename='gesture_detection.log',
5     level=logging.INFO,
6     format='%(asctime)s - %(levelname)s - %(message)s'
7 )

```

Minden értesítés kiküldésénél(`notify_subscribers()` függvénybe) beleírtam egy log fájlba a felismert gesztus nevét, és egy változóval ellenőriztem, hogy ne fusson ki a 30 próbából.

Statikus gesztus

30-szor formáztam a kezemmel „Zoom In” gesztust, amiből 22-szer eltalálta a rendszer. A 6.4. ábrán látható a log fájlban az eredmény.



```

gesture_detection.log
Reveal Now Clear Reload Share
2025-04-16 20:26:23,281 - INFO - Gesture detected: None
2025-04-16 20:26:23,427 - INFO - Gesture detected: None
2025-04-16 20:26:24,784 - INFO - Gesture detected: None
2025-04-16 20:26:25,002 - INFO - Gesture detected: None
2025-04-16 20:26:25,164 - INFO - Gesture detected: Zoom In
2025-04-16 20:26:25,273 - INFO - Gesture detected: Zoom In
2025-04-16 20:26:25,971 - INFO - Gesture detected: Zoom In
2025-04-16 20:26:26,136 - INFO - Gesture detected: Zoom In
2025-04-16 20:26:26,302 - INFO - Gesture detected: Zoom In
2025-04-16 20:26:27,236 - INFO - Gesture detected: Zoom In
2025-04-16 20:26:28,325 - INFO - Gesture detected: Zoom In
2025-04-16 20:26:29,231 - INFO - Gesture detected: None
2025-04-16 20:26:29,462 - INFO - Gesture detected: Zoom In
2025-04-16 20:26:29,629 - INFO - Gesture detected: Zoom In
2025-04-16 20:26:30,686 - INFO - Gesture detected: Zoom In
2025-04-16 20:26:31,657 - INFO - Gesture detected: Zoom In
2025-04-16 20:26:32,745 - INFO - Gesture detected: None
2025-04-16 20:26:32,909 - INFO - Gesture detected: Zoom In
2025-04-16 20:26:35,930 - INFO - Gesture detected: None
2025-04-16 20:26:36,102 - INFO - Gesture detected: None
2025-04-16 20:26:36,266 - INFO - Gesture detected: Zoom In
2025-04-16 20:26:37,057 - INFO - Gesture detected: Zoom In
2025-04-16 20:26:37,361 - INFO - Gesture detected: Zoom In
2025-04-16 20:26:38,371 - INFO - Gesture detected: Zoom In
2025-04-16 20:26:38,662 - INFO - Gesture detected: Zoom In
2025-04-16 20:26:39,457 - INFO - Gesture detected: Zoom In
2025-04-16 20:26:39,624 - INFO - Gesture detected: Zoom In
2025-04-16 20:26:40,437 - INFO - Gesture detected: Zoom In
2025-04-16 20:26:40,607 - INFO - Gesture detected: Zoom In
2025-04-16 20:26:41,396 - INFO - Gesture detected: Zoom In
2025-04-16 20:26:41,400 - INFO - Received 30 gestures, stopping logging. 'Zoom In' gestures count: 22

```

6.4. ábra. A log fájl tartalma statikus gesztus esetén

Dinamikus gesztus

Ebben az esetben is 30-szor próbálkoztam, csak most a „Swipe Right” gesztussal. Itt is hasonló pontosságú a felismerés, ahogy a lenti ábrán is látható, 30-ból 20-szor ismerte fel a gesztust, ahogy a 6.5. ábrán is látható a log fájl.

```

gesture_detection.log
Reveal Now Clear Reload Share
2025-04-17 18:22:57,617 - INFO - Gesture detected: None
2025-04-17 18:22:58,438 - INFO - Gesture detected: Swipe Right
2025-04-17 18:23:00,814 - INFO - Gesture detected: Swipe Left
2025-04-17 18:23:02,514 - INFO - Gesture detected: None
2025-04-17 18:23:02,891 - INFO - Gesture detected: Swipe Right
2025-04-17 18:23:04,912 - INFO - Gesture detected: None
2025-04-17 18:23:05,213 - INFO - Gesture detected: Swipe Right
2025-04-17 18:23:05,384 - INFO - Gesture detected: Swipe Right
2025-04-17 18:23:06,981 - INFO - Gesture detected: None
2025-04-17 18:23:07,427 - INFO - Gesture detected: Swipe Right
2025-04-17 18:23:09,661 - INFO - Gesture detected: Swipe Right
2025-04-17 18:23:09,830 - INFO - Gesture detected: Swipe Right
2025-04-17 18:23:09,921 - INFO - Gesture detected: Swipe Right
2025-04-17 18:23:12,105 - INFO - Gesture detected: Swipe Right
2025-04-17 18:23:12,379 - INFO - Gesture detected: Swipe Right
2025-04-17 18:23:14,440 - INFO - Gesture detected: Swipe Left
2025-04-17 18:23:16,407 - INFO - Gesture detected: Swipe Right
2025-04-17 18:23:16,595 - INFO - Gesture detected: Swipe Right
2025-04-17 18:23:20,616 - INFO - Gesture detected: Zoom In
2025-04-17 18:23:20,933 - INFO - Gesture detected: Swipe Right
2025-04-17 18:23:21,057 - INFO - Gesture detected: Swipe Right
2025-04-17 18:23:23,208 - INFO - Gesture detected: Swipe Left
2025-04-17 18:23:23,388 - INFO - Gesture detected: Swipe Right
2025-04-17 18:23:29,744 - INFO - Gesture detected: Swipe Right
2025-04-17 18:23:29,986 - INFO - Gesture detected: Swipe Right
2025-04-17 18:23:31,920 - INFO - Gesture detected: Swipe Left
2025-04-17 18:23:32,098 - INFO - Gesture detected: Swipe Right
2025-04-17 18:23:38,433 - INFO - Gesture detected: Swipe Right
2025-04-17 18:23:38,592 - INFO - Gesture detected: Swipe Right
2025-04-17 18:23:40,265 - INFO - Gesture detected: Swipe Left
2025-04-17 18:23:40,270 - INFO - Received 30 gestures, stopping logging. 'Swipe Right' gestures count: 20

```

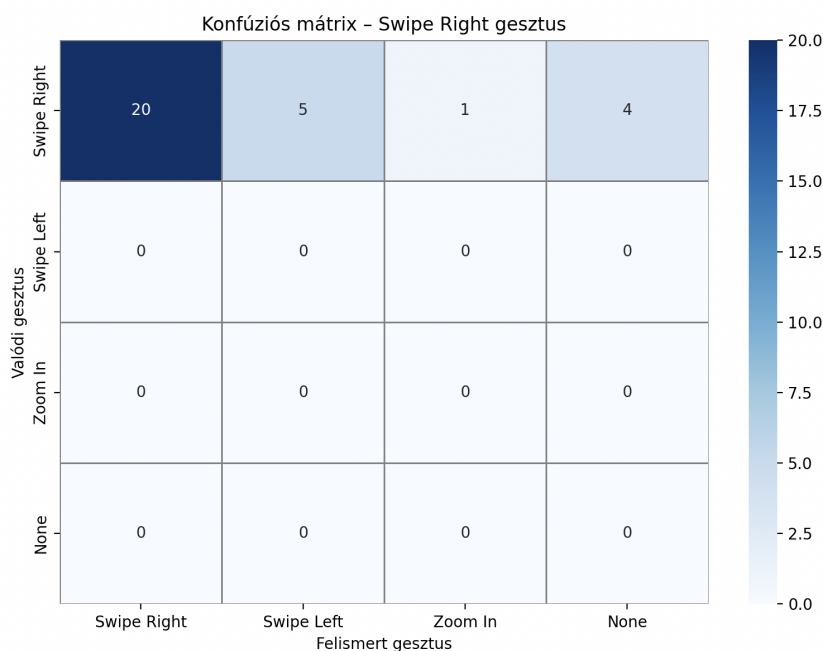
6.5. ábra. A log fájl tartalma dinamikus gesztus esetén

Konfúziós mátrix

A további elemzéshez egy konfúziós mátrixot készítettem (lásd 6.6. ábra), amely bemutatja, hogyan teljesített a rendszer a „Swipe Right” dinamikus gesztus felismerésénél. A 30 próbálkozás során minden esetben a „Swipe Right” gesztust mutattam be, azonban a rendszer az alábbi felismeréseket adta:

- 20 alkalommal helyesen ismerte fel a „Swipe Right” gesztust
- 5 alkalommal tévesen „Swipe Left” gesztust ismert fel
- 4 alkalommal nem tudott gesztust felismerni („None”)
- 1 alkalommal a „Zoom In” gesztust ismerte fel

Ez azt mutatja, hogy a modell a nem felismert esetek többségében egy másik, irányban ellentétes mozdulattal („Swipe Left”) keverte össze az eredeti gesztust, ami arra utal, hogy a rendszer érzékeny az irányváltások finom különbségeire. A 6.6. ábra szemlélteti ezt a hibamegoszlást.



6.6. ábra. Konfúziós mátrix a „Swipe Right” gesztus felismeréséről

Pontosság

A gesztusfelismerés pontosságát a következő képlettel számítottam:

$$\text{Pontosság} = \frac{\text{Helyesen felismert esetek száma}}{\text{Összes próbálkozás száma}} \times 100\%$$

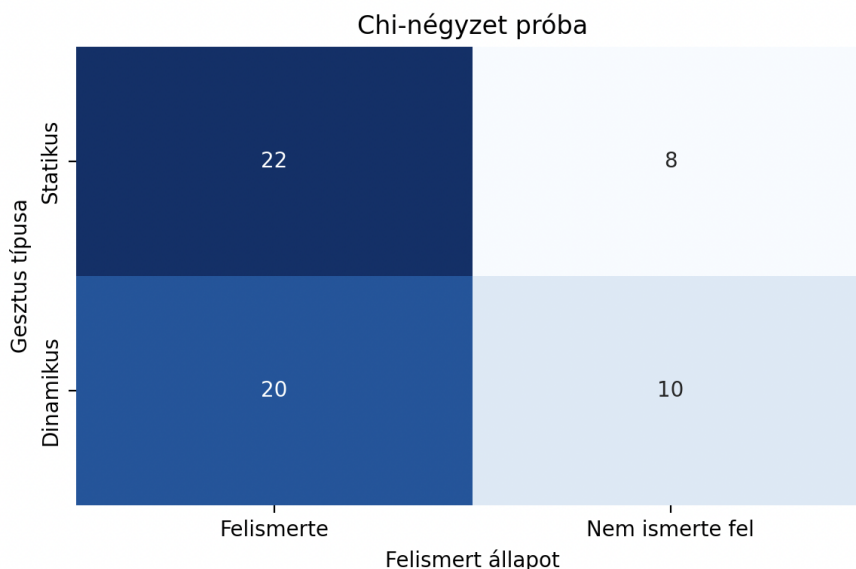
- Statikus gesztus: $\frac{22}{30} \times 100\% = 73,3\%$
- Dinamikus gesztus: $\frac{20}{30} \times 100\% = 66,7\%$

A két vizsgált gesztus („Zoom In” – statikus, illetve „Swipe Right” – dinamikus) felismerési pontosságának összehasonlítására **Chi-négyzet próbát** alkalmaztam (lásd 6.7. ábra). Ez a teszt azt vizsgálja, hogy van-e statisztikailag szignifikáns különbség két kategóriaváltozó között. Jelen esetben a gesztus típusa és annak felismerése között.

A próbához az alábbi megfigyelési eredményeket használtam:

Gesztus típusa	Felismerte	Nem ismerte fel
Státikus (Zoom In)	22	8
Dinamikus (Swipe Right)	20	10

A teszt elvégzése után a kapott $\chi^2 = 0,079$ és p-érték 0,778 volt, ami magasabb a szokásos 0,05-ös szignifikanciaszintnél. Ez azt jelenti, hogy a megfigyelt különbség a felismerések arányában nem tekinthető statisztikailag szignifikánsnak, vagyis nincs elegendő bizonyíték arra, hogy a két gesztus felismerési arányai között valódi különbség van.



6.7. ábra. Chi-négyzet próba a felismerések összehasonlítására

6.3. Fényviszonyok

A következő mérési szempont a fényviszonyok voltak. Mértém a szegmentálás romlását a fény csökkenésével, és a gesztus felismerést pontosságát. A fény mérésére a mobiltelefonomat (Iphone 15 pro) használtam, aminek a kamerájával, egy ingyenesen letölthető alkalmazás (Lux Light Meter Pro for Mobile) segítségével mértem a megvilágítás erősségét, **lux** mértékegységben. A lux a megvilágítás mértékegysége, vagyis azt mutatja meg, hogy egy adott felületre mennyi fény érkezik.

$$1 \text{ lux} = 1 \text{ lumen/m}^2$$

A 6.8. ábrán látható a Lux Light Meter Pro for Mobile alkalmazás, jelen esetben egy szobában mértem a megvilágítást. Természetes nappali fény és lámpa is megvilágított a mérés során.



6.8. ábra. A telefonos alkalmazással mért lux mennyiség, szobában természetes fény és lámpa megvilágítással

Az alábbi képeken mutatom be a mérések eredményét szegmentálás és gesztus felismerés során.

A 6.9. ábrán természetes fényt használtam a délelőtti órákban. Kissé felhős idő volt. **3120 lux** volt a megvilágítás. Ebben az esetben látszik, hogy nagyon pontos a szegmentálás és a gesztust is helyesen felismeri.



6.9. ábra. Természetes fényben mért megvilágítás

A 6.10. ábrán látható, hogy ha nem a szabadban levő fény világít meg, hanem egy szobában ülve a betni, plusz a kintről beszivárgó fény süt rám, akkor kicsit romlik a szegmentálás minősége, de még így is elfogadható. Ekkor **289 lux**-ot mértem.



6.10. ábra. Szobában mért megvilágítás

A 6.11. ábrán már kintről is alig szűrődött be fény, és lámpák se világítottak. Ebben az esetben **78 lux**-ot mért a telefonom. Itt már látható a szegmentálás romlása, zavaró hogy nem tudja elkülöníteni a kezem a háttértől egy egy helyen. A gesztus felismerése még mindid pontos, ilyen megvilágítások között is.



6.11. ábra. Gyenge fényviszonyok

Sötétben, úgy, hogy csak a laptop képernyője világít meg, drasztikusan romlik a szegmentálás. A 6.12. ábrán látható, hogy szinte teljesen sötét az emberi alak, bár valamilyen szinten mégis el tudta különíteni a háttértől. Ebben az esetben a kéz

szegmentálása még rosszabb, alig lehet kivenni a kéz formáját. Ilyen fényviszonyok között a mért megvilágítás **5 lux** volt.



6.12. ábra. Sötétben a szegmentálás romlása

6.4. Erőforrások

A kutatás során a saját laptopomat, egy 2017-es MacBook Pro-t használtam, amelyen az API, a mesterséges intelligencia modellek, valamint a prezentációs alkalmazás felhasználói felülete is futott. A laptop specifikációi a következők:

- **CPU:** 2,8 GHz Quad-Core Intel Core i7
- **RAM:** 16 GB
- **GPU:** Radeon Pro 555 (2 GB) és Intel HD Graphics 630

A legnagyobb kihívást a valós idejű képszegmentálás jelentette. Mivel a legtöbb AI-modell futtatásához nagy számítási kapacitás szükséges (pl. CUDA-maggal rendelkező NVIDIA GPU), a választás végül a **MediaPipe** modellre esett. A kutatás elején több más szegmentáló megoldást is kipróbáltam, azonban mivel a saját gépemben nincs CUDA-támogatás, alternatívát kellett keresnem.

Kipróbáltam a Sapientia EMTE Marosvásárhelyi karán található Rocks szervert, amelyhez SSH tunelen keresztül csatlakoztam. A laptopom kameraképét továbbítottam a szerver felé, ahol megtörtént a feldolgozás. A Rocks szerver egy NVIDIA GTX 1070 GPU-val rendelkezik, amely bár rendelkezik CUDA-maggal, a valós idejű működéshez még így is lassúnak bizonyult.

Emellett kipróbáltam még a Google Colab-ot is. Itt a T4 GPU-t használtam a Meta Sapiens szegmentáló modelljéhez. Egy feltöltött kép fájlt 3-4 másdperc alatt

szegmentált, ami a saját laptopomhoz (egy képfájl szegmentálása a Sapiens-el 50-60 másodperc volt) képest jobb eredmény volt. Mivel a Colab a böngészőben érhető el, így nem tudtam valós idejűségre használni, vagyis nem tudtam a laptopom kameráját használva valós időben szegmentálásra használni.

Ezek után döntöttem a MediaPipe mellett, amely meglepően gyorsan és pontosan működött, különösen valós idejű képszegmentálásnál. Hátrányként azonban megemlíthető, hogy gyenge fényviszonyok között nem annyira pontos.

A 6.1. táblázat az egyes modellek képszegmentálás közbeni teljesítményét mutatja, képkocka/másodperc (FPS) értékek alapján, különböző környezetekben mérve:

Modell neve és futtatási környezet	FPS
Meta Sapiens (saját laptop)	0.002
Meta Sapiens (Rocks szerver)	0.02
YOLOv8 (saját laptop)	2
YOLOv8 (Rocks szerver)	7
MediaPipe (saját laptop)	15-20

6.1. táblázat. Különböző modellek teljesítménye képszegmentálás során

7. fejezet

Eredmények tárgyalása

A kutatás célkitűzése az volt, hogyan valósítható meg egy rugalmas, valós idejű gesztusfelismerő API, amely integrálható különböző platformokra, különös tekintettel az interaktív oktatási és prezentációs környezetekre. Ennek megfelelően a mérések négy fő szempont köré szerveződtek: válaszidő (latencia), gesztusfelismerés pontossága, fényviszonyokra való érzékenység, valamint az erőforrásigény és a futtathatóság különböző környezetekben. Az alábbiakban ezen mérések eredményeit értékelem és összegzem.

7.1. Latencia elemzése

A kliens és az API közötti kommunikáció válaszideje kulcsfontosságú tényező egy valós idejű gesztusvezérlésű rendszer működésében. A 100 mintából végzett mérés alapján az API átlagos válaszideje 276 ms volt, amely technikai szempontból megfelelőnek tekinthető. Ugyanakkor a viszonylag magas, 202,22 ms-os szórás arra utal, hogy az API válaszideje nem teljesen stabil. A hisztogram elemzése azt mutatta, hogy az esetek több mint fele (52%) a leggyorsabb tartományba esett (124–202,22 ms), vagyis a rendszer legtöbbször megfelelően gyors választ adott. Ez biztató a valós idejű alkalmazás szempontjából, ugyanakkor a rendszer stabilitásának javítása tovább növelné a megbízhatóságot.

7.2. Gesztusfelismerés pontossága

A mérések során kétféle gesztustípust vizsgálok: egy statikus („Zoom In”) és egy dinamikus („Swipe Right”) mozdulatot. A mérési eredmények szerint a statikus gesztusokat nagyobb pontossággal ismerte fel a rendszer (73,3%), míg a dinamikus gesztusok esetén ez az arány csökkent (66,7%).

A dinamikus gesztusok pontosságának csökkenése érthető, hiszen az időbeli változásokat nehezebb megbízhatóan érzékelni és feldolgozni, főleg egy olyan rendszer esetében, amelynek nincs dedikált, mély tanuláson alapuló időbeli követőmechanizmus. A konfúziós mátrix elemzése lapján az eltévesztett felismerések többsége (5 eset) „Swipe Left” mozdulatként került beazonosításra, ami arra utal, hogy a rendszer nehezen különbözteti meg a véletlen mozdulatokat. Ez fejlesztési lehetőséget kínál az irányérzékenység növelésére.

A Chi-négyzet próba eredménye ($p = 0,079$) szerint a pontosság különbsége nem tekinthető statisztikailag szignifikánsnak, így jelen mintán nem tudjuk egyértelműen kijelenteni, hogy a típus befolyásolja a felismerés sikerességét. A minta bővítésével azonban ez az eredmény változhat.

7.3. Fényviszonyokra való érzékenység

A mérésekből jól látszik, hogy a megvilágítás szintje jelentős hatással van a szegmentálás minőségére, de a gesztusfelismerés megbízhatóságára nem. Közepes vagy erős megvilágítás mellett (3000 lux környékén) a rendszer szinte hibátlanul működött, míg gyenge fényviszonyoknál (78 lux alatt) a szegmentálás pontossága romlott. Sötétben (5 lux) a kéz és a háttér közötti kontraszt szinte eltűnt, de még ebben az esetben is a gesztus felismerés pontos. Ugyanakkor ezek az eredmények rávilágítanak arra, hogy a kép szegmentálás jelentősen függ a környezeti megvilágítástól, így beltéri, esti vagy alacsony fényviszonyok melletti használat esetén kiegészítő világításra lehet szükség.

7.4. Erőforrásigény és teljesítmény

A különböző szegmentáló modellek összehasonlítása alapján egyértelmű, hogy a MediaPipe választása indokolt volt. A saját laptopomon futtatott MediaPipe modell 15-20 FPS-sel működött, amely megfelelő gördülékenységet biztosított a valós idejű működéshez. Ezzel szemben más, komplexebb modellek (pl. Meta Sapiens vagy YOLOv8) jelentősen lassabbak voltak, különösen a saját gépen, CUDA-támogatás hiányában.

A Rocks szerver használata sem hozott áttörést, ugyanis bár erősebb hardverrel rendelkezik, a valós idejű működéshez még így is kevés volt a feldolgozási sebesség. A MediaPipe tehát nemcsak pontos gesztus felismerés szempontjából, de erőforrás-hatékony választásnak is bizonyult.

A MediaPipe-alapú rendszer megfelelő kiindulási állapot jelentett egy valós idejű rendszer megvalósítására, különösen akkor, ha a fényviszonyok kedvezőek. A jövőbeni fejlesztések során érdemes lenne a dinamikus gesztusok felismerését tovább javítani, valamint az alacsony fényviszonyok melletti működés stabilitását növelni.

8. fejezet

Következtetések

A kutatás során a cél az volt, hogy megoldást találjunk arra a kérdésre, hogy miként valósítható meg egy rugalmas, valós idejű gesztusfelismerő API, amely webhookok segítségével hatékonyan integrálható különböző alkalmazásokba, különös tekintettel az interaktív prezentációs környezetekre. Az elvégzett fejlesztések és tesztelések alapján az alábbi következtetések vonhatók le.

A rugalmas gesztusfelismerő API sikeres megvalósítása érdekében kulcsfontosságú volt a megfelelő technológiai alapok választása. Az API a MediaPipe könyvtárat használta [17], amely jól optimalizált és könnyen integrálható különböző platformokon. Az API működése során a kliens folyamatosan képkockákat küldött egy endpointra, ahol a gesztusokat mesterséges intelligencia modellel dolgoztuk fel. A webhook-alapú értesítések valós idejű kommunikációt biztosítottak, lehetővé téve az eseményvezérelt interakciókat. Ezen keresztül a rendszer könnyen integrálható volt különböző alkalmazásokba, és biztosította az interaktív prezentációs környezetekben szükséges dinamikus reakciókat.

A kutatás eredményei alapján az API válaszideje 276 ms körüli volt, ami megfelelőnek bizonyult a valós idejű alkalmazásokhoz [11]. Azonban a tesztelés során megfigyeltük, hogy a válaszidő stabilitása és az erőforrás-hatékonyság javítására még van lehetőség, különösen dinamikus a gesztusok esetén. A fényviszonyok változása ugyanakkor hatással volt a rendszer teljesítményére. Sötét környezetben a szegmentálás pontossága csökkent, de a rendszer általánosan jól működött különböző fényviszonyok mellett is.

A tesztelés során kiemelkedett az API rugalmassága és alkalmazhatósága különböző területeken. A rendszer jól alkalmazható volt interaktív prezentációkban [24], ahol a felhasználó gesztusai közvetlen hatással voltak a prezentált tartalomra. Ezen kívül az API könnyen adaptálható egy COVID-19 pandémiához [4] hasonló helyzetben, lehetővé téve az érintésmentes interakciókat az online oktatásban. A háromnyelvű felhasználói felület tovább növelte az API hozzáférhetőségét nemzetközi szinten.

Összességében a kutatás eredményei azt mutatják, hogy egy rugalmas, valós idejű gesztusfelismerő API, amely webhookok segítségével integrálható különböző alkalmazásokba, hatékony megoldást kínálhat interaktív prezentációs és oktatási környezetekben, miközben biztosítja a felhasználók számára a gördülékeny és dinamikus élményt. De ezek mellett számos dologra is fel lehet használni az API-t, az okosotthonok [6], a művészet, az autóipar [21] területén, vagy akár az egészségügyben [22].

8.1. Továbbfejlesztési lehetőségek

A kutatás egy másik célja a rendszer folyamatos fejlesztése volt, különösen a dinamikus gesztusok felismerésének javítása. A jövőbeli fejlesztések során egy nagyobb erőforrásokkal rendelkező számítógépen szeretném futtatni az API-t, hogy tudjak pontosabb szegmentáló modellt használni (pl. Meta Sapiens). Ezek mellett a fényviszonyok melletti stabilitás növelésére összpontosítanék, hogy az API még szélesebb körben és különböző környezetekben alkalmazható legyen. Továbbá az alkalmazás (az API és az oktatást segítő platform) kitelepítése növelné a felhasználók számát. A több felhasználóval lehetne egy felmérést csinálni, hogy melyek azok az intuitív gesztusok, amikkel szeretnének használni az alkalmazás során, de jelenleg hiányoznak belőle. Ezáltal bővíteném a felismert gesztusok számát.

Az oktatást segítő prezentáció alkalmazást szeretném elérhetővé tenni mobil felületre is. Így többen és könnyebben használnák az emberek, mert nem kell egy számítógép mellett kövessék a tanórát. Az alkalmazásba bele szeretném vinni a szemmel való irányítást is (eye-tracking), felhasználói szerepköröket. Regisztrációhoz kötném a felhasználást, de megmarada a vendég felhasználó is. A regisztrációval lenne saját profilja a felhasználónak, így a gesztus - funkció beállításai nem csak a munkamenet végéig (ablak bezárása) mentődnének el. A felhasználók megjelenésével bevezetném az arc felismerést, ami megkönnyítené a fiókba való belépést, majd a prezentáció közbeni használatot is, hiszen ha többen vannak egymás mellett, akkor csak annak a személynek ismerné fel a gesztusát aki be van jelentkezve.

Irodalom

- [1] Boštjan Šumak, Saša Brdnik és Maja Pušnik. «Sensors and artificial intelligence methods and algorithms for human–computer intelligent interaction: A systematic mapping study». *Sensors* 22.1 (2021), 20. old. (hiv. old. 1).
- [2] All About AI. *Gesture Recognition*. 2024. URL: <https://www.allaboutai.com/ai-glossary/gesture-recognition> (hiv. old. 1).
- [3] The Tech Vortex. *Gesture Recognition Technology Pros*. 2024. URL: <https://the-tech-vortex.com/2024/11/12/gesture-recognition-technology-pros> (hiv. old. 1).
- [4] Balázs Adorján. «A digitális oktatás tapasztalatai a COVID-19 járvány idején». *Képzés és Gyakorlat: Neveléstudományi folyóirat* 18.3-4 (2020), 213–226. old. (hiv. old. 2, 65).
- [5] Tian Zhang. «Application of AI-based real-time gesture recognition and embedded system in the design of English major teaching». *Wireless Networks* (2021), 1–13. old. (hiv. old. 2).
- [6] Jingwei Dai. *Gesture Recognition Based Smart Home Control System*. 2020. URL: <https://ojs.bbwpublisher.com/index.php/JERA/article/view/1057> (hiv. old. 2, 66).
- [7] Anupama A, Ramya S Yamikar és Renuka K M. «AI-driven gesture control for industrial robots: A vision-based approach enhancing human-robot collaboration». *World Journal of Advanced Research and Reviews* (2023). URL: <https://wjarr.com/sites/default/files/WJARR-2023-2254.pdf> (hiv. old. 3).
- [8] Richard Szeliski. *Computer vision: algorithms and applications*. Springer Nature, 2022 (hiv. old. 4).
- [9] Yadira Quiñonez, Carmen Lizarraga és Raquel Aguayo. «Machine learning solutions with MediaPipe». *2022 11th International Conference On Software Process Improvement (CIMPS)*. IEEE. 2022, 212–215. old. (hiv. old. 4, 14).

- [10] Zhe Cao, Tomas Simon, Shih-En Wei és Yaser Sheikh. «Realtime multi-person 2d pose estimation using part affinity fields». *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2017, 7291–7299. old. (hiv. old. 5).
- [11] Joseph Redmon, Santosh Divvala, Ross Girshick és Ali Farhadi. «You only look once: Unified, real-time object detection». *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, 779–788. old. (hiv. old. 5, 65).
- [12] Pavlo Molchanov, Xiaodong Yang, Shalini Gupta, Kihwan Kim, Stephen Tyree és Jan Kautz. «Online detection and classification of dynamic hand gestures with recurrent 3d convolutional neural network». *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, 4207–4215. old. (hiv. old. 5).
- [13] Sijie Yan, Yuanjun Xiong és Dahua Lin. «Spatial temporal graph convolutional networks for skeleton-based action recognition». *Proceedings of the AAAI conference on artificial intelligence*. 32. köt. 1. 2018 (hiv. old. 5).
- [14] Alexey Dosovitskiy és tsai. «An image is worth 16x16 words: Transformers for image recognition at scale». *arXiv preprint arXiv:2010.11929* (2020) (hiv. old. 6).
- [15] Ian Goodfellow, Yoshua Bengio, Aaron Courville és Yoshua Bengio. *Deep learning*. 1. köt. 2. MIT press Cambridge, 2016 (hiv. old. 6).
- [16] Abolfazl Younesi, Mohsen Ansari, Mohammadamin Fazli, Alireza Ejlali, Muhammad Shafique és Jörg Henkel. «A comprehensive survey of convolutions in deep learning: Applications, challenges, and future trends». *IEEE Access* 12 (2024), 41180–41218. old. (hiv. old. 7).
- [17] Google. *MediaPipe Hands*. <https://mediapipe.readthedocs.io/en/latest/solutions/hands.html>. 2024 (hiv. old. 8, 65).
- [18] Qualcomm AI Hub - MediaPipe Hand Model. https://aihub.qualcomm.com/models/mediapipe_hand. 2023 (hiv. old. 9).
- [19] Alice Heiman. *Recognize Hand Landmarks using Google MediaPipe and OpenCV*. <https://medium.com/@aliceheimanxyz/recognize-hand-landmarks-using-google-mediapipe-and-opencv-9ca0a052ce75>. 2023 (hiv. old. 9).
- [20] Jun Tian, Weilie Zhang, Tao Zhang, Guopeng Li és Zhang Long. «Research status of gesture recognition based on vision: A review». *IOP Conference Series: Earth and Environmental Science*. 632. köt. 4. IOP Publishing. 2021, 42019. old. (hiv. old. 9, 14).

- [21] Zaiyan Gong. «The History of Automotive Human-Machine Interaction: Seven Directions of Evolution». *International Conference on Human-Computer Interaction*. Springer. 2025, 36–50. old. (hiv. old. 10, 66).
- [22] A-reum Lee, Yongwon Cho, Seongho Jin és Namkug Kim. «Enhancement of surgical hand gesture recognition using a capsule network for a contact-less interface in the operating room». *Computer methods and programs in biomedicine* 190 (2020), 105385. old. (hiv. old. 10, 66).
- [23] Hosam Al-Samarraie és Nasir Saeed. «The impact of pandemic-driven remote learning on interaction and collaboration in online education: A review». *Computers & Education* 159 (2020), 104034. old. DOI: 10.1016/j.compedu.2020.104034 (hiv. old. 10, 14).
- [24] Sunghee Lee, Heecheol Kim és Wonjun Lee. «Augmented reality based on Kinect sensor for interactive e-learning». *2013 IEEE International Conference on Consumer Electronics (ICCE)*. IEEE. 2013, 252–253. old. DOI: 10.1109/ICCE.2013.6486930 (hiv. old. 11, 14, 65).
- [25] Javier Ruiz, Guillermo García és Inmaculada Rodríguez. «Gesture-based interaction in assistive technology: A study on accessibility and usability for people with motor disabilities». *International Journal of Human-Computer Interaction* 36.16 (2020), 1525–1541. old. DOI: 10.1080/10447318.2020.1781263 (hiv. old. 11, 14).
- [26] Chi-Hung Chen és Yu-Hsin Tsai. «Game-based learning and gesture interaction for physics learning in a virtual laboratory». *Interactive Learning Environments* 25.3 (2017), 343–357. old. DOI: 10.1080/10494820.2016.1138313 (hiv. old. 11, 15).
- [27] Krrish Naik. *SelfieSegmentation_video*. https://github.com/imkrrish/SelfieSegmentation_video. GitHub repository. 2021 (hiv. old. 37).