Вінницький національний технічний університет

Факультет інформаційних технологій та комп'ютерної інженерії

Кафедра програмного забезпечення

Пояснювальна записка

до магістерської кваліфікаційної роботи

магістр
(освітньо-кваліфікаційний рівень)

на тему: Розробка методів і програмного забезпечення доповненої реальності для розпізнавання рухів з використанням технологій Swift, ARKit, CoreML.

Виконав: студент 2 курсу, групи 2ПІ-18м
напряму підготовки
121 – Інженерія програмного забезпечення
(шифр і назва напряму підготовки, спеціальності)

Туйчев Владислав Володимирович
(прізвище та ініціали)

Керівник: к.т.н., доц. каф. ПЗ Катєльніков Д.І.
(прізвище та ініціали)

Рецензент: к.т.н., доц. каф. КН Арсенюк І.Р.
(прізвище та ініціали)

ВНТУ – 2019

Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії
Кафедра програмного забезпечення
Освітньо-кваліфікаційний рівень – магістр
Спеціальність 121 – Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ
Зав. кафедри ПЗ
д.т.н., проф. Романюк О.Н.
_____
«_____»_____2019 р.

ЗАВДАННЯ
НА МАГІСТЕРСЬКУ КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТУ
Туйчеву Владиславу Володимировичу

1. Тема магістерської дипломної роботи: «Розробка методів і програмного забезпечення доповненої реальності для розпізнавання рухів з використанням технологій Swift, ARKit, CoreML».

Керівник магістерської дипломної роботи: Катєльніков Д.І. , к.т.н., доцент кафедри ПЗ, затверджені наказом вищого навчального закладу від "__"_____2019 року №___

2. Строк подання студентом роботи _____

3. Вихідні дані до роботи:

Мова програмування: Swift;

Середовище розробки: Xcode;

Тип нейронної мережі: Згорткова нейронна мережа;

Бібліотеки: Swift, ARKit, CoreML, PyTorch.

Зміст розрахунково-пояснювальної записки: вступ; аналіз сучасного стану вирішення задачі та постановка завдання; розробка методу інтерактивної

взаємодії користувач-комп’ютер; розробка мобільного додатку; тестуван додатку; економічна частина; висновки; перелік посилань; додатки.

5. Перелік графічного матеріалу (з точним зазначенням обов’язкових креслень): тема, автор, науковий керівник; актуальність; мета, об'єкт та предмет дослідження; задачі; наукова новизна; практичне значення; порівняння з аналогами; алгоритм знаходження найвищої точки; технології; тестування програми; ресурсозатразтність; апробація матеріалів.

6. Консультанти розділів магістерської кваліфікаційної роботи

Розділ	Прізвище, ініціали та посада Консультанта	Підпис, дата	
		завдання видав	завдання прийняв
1-4	к.т.н. Катєльніков Д.І. доцент. кафедри ПЗ		
5	к.е.н. Бальзан М. В., доцент кафедри ЕПВМ		

7. Дата видачі завдання_____

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів магістерської кваліфікаційної роботи	Строк виконання етапів роботи	Примітка
1	Аналіз стану проблеми та порівняльний аналіз аналогів	01.10.19 – 11.10.19	Вик.
2	Розробка методів інтерактивної взаємодії користувач-комп’ютер	12.10.19 – 17.10.19	Вик.
3	Розробка програмного забезпечення	18.10.19 – 27.11.19	Вик.
4	Тестування програмної системи	28.11.19 – 30.11.19	Вик.
5	Економічна частина	01.12.19 – 08.12.19	Вик.

Студент _____ **Туйчев В.В.**
(підпис) (прізвище та ініціали)

Керівник магістерської кваліфікаційної роботи _____ **Катєльніков Д. І.**
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Магістерська кваліфікаційна робота присвячена розробці програмної системи на мобільні пристрої для розпізнавання рухів. Інтерфейс дозволяє взаємодіяти з об’єктами доповненої реальності за допомогою руки користувача.

Запропонована нова модель взаємодії користувача з віртуальними об’єктами в реальному світі.

Програмний додаток був реалізований засобами мови програмування Swift в середовищі програмування Xcode, а також мови програмування Python для навчання нейронної моделі. Технологія CoreML використовує навчену модель, та дозволяє розпізнавати людську руку, за допомогою ARKit користувач може створити віртуальний об’єкт та взаємодіяти з ним за допомогою руки.

ABSTRACT

Master's qualification work is devoted to the development of a software system for mobile devices for motion recognition. The interface allows you to interact with augmented reality objects with the help of the user's hand.

A new model of user interaction with virtual objects in the real world is proposed.

The software was implemented using the Swift programming language in the Xcode programming environment, as well as Python programming language to train the neural model. CoreML technology uses a trained model to identify the human hand, using ARKit to create and interact with a virtual object.

ЗМІСТ

ВСТУП

Обґрунтування вибору теми дослідження. Сьогодні важко уявити життя в світі без комп’ютерів та смартфонів. Завдяки сучасним технологіям люди мають нові робочі місця, нові види розваг та відпочинку, новий рівень освіти та безпеки, лікування і т.д.

Смартфони – окрема категорія телефонів, які на відміну від простих стільникових телефонів мають більше оперативної пам'яті і власний потужний, як для кишенькових пристроїв процесор, працюють під операційними системами iOS, Android та інших [1]. У світі близько 2 млрд користувачів смартфонів і з кожним днем вони відіграють все більшу роль у людському житті. Спочатку це був пристрій для зв’язку на далеких відстанях, зараз – медіатека, мессенджери, ігри, програми, керування банківськими рахунками, оплата товарів, фотоапарат, навігатор і т.д.

Можливості використання смартфонів постійно зростають. Так наприклад компанія Apple додала в iOS пристрої функцію BLE[2] і розглянула її на прикладі використання смартфону як безконтактного ключа від дверей в будинок. Але не менш цікавим є способи розпізнавання людських рухів.

Розпізнавання жестів – одна з головних задач комп’ютерних наук та технологій з метою інтерпретації людських жестів за допомогою математичних алгоритмів. Жести можуть створюватися завдяки руху тіла або його певного стану. Люди можуть використовувати прості жести для керування або взаємодії з пристроями, не торкаючись їх фізично. Багато підходів було зроблено за допомогою камер та алгоритмів комп'ютерного зору для інтерпретації мови жестів. Використовуючи концепцію розпізнавання жестів, можна керувати пристроєм лише вказівним пальцем. Це допоможе максимально зменшити введення даних за механічним способом.

Отже, питання розпізнавання рухів та жестів є досить важливим в сучасному світі, адже це не лише спрощує спосіб введення даних, а й робить цей

процес швидшим. Саме тому розробка методів і програмного забезпечення доповненої реальності для розпізнавання рухів є актуальною.

Мета та завдання дослідження. Метою роботи є покращення взаємодії користувач-комп’ютер за рахунок поєднання методів створення доповненої реальності і розпізнавання рухів.

Основними задачами дослідження є:

- провести аналіз існуючих методів і засобів створення доповненої реальності для визначення напрямків їх ефективного використання з системою розпізнавання рухів;
- запропонувати нові:
    a) методи інтерактивної взаємодії з урахуванням розпізнавання жестів рук, положення долоні, кутів суглобів тощо;
    b) методи поєднання доповненої реальності і розпізнавання рухів;
- розробити програмні компоненти та систему доповненої реальності на основі запропонованих методів;
- провести експериментальні дослідження розроблених засобів інтерактивної взаємодії користувач-комп’ютер на основі доповненої реальності та розпізнавання жестів рук, положення долоні, кутів суглобів.

Об’єкт дослідження – процес інтерактивної взаємодії користувач-комп’ютер.

Предмет дослідження – методи та засоби розпізнавання жестів рук.

Зв’язок роботи з науковими програмами, планами, темами. Робота виконувалась згідно плану виконання наукових досліджень на кафедрі програмного забезпечення.

Методи дослідження. У процесі дослідження використовувались теорія штучного інтелекту, теорія машинного навчання, теорія розпізнавання образів, теорія комп’ютерного зору для розробки методу інтерактивної взаємодії з врахуванням розпізнавання жестів рук, положення долоні, кутів суглобів, комп’ютерне моделювання для аналізу та перевірки отриманих теоретичних положень.

Наукова новизна отриманих результатів.

1. Подальшого розвитку отримав метод інтерактивної взаємодії користувач-комп’ютер у якому, на відміну від існуючих, використано не тільки дані маніпуляторів миша та клавіатура, але й враховується результати розпізнавання жестів рук, положення долоні, кутів суглобів, що дозволяє значно розширити діапазон можливих команд.
2. Подальшого розвитку отримав метод формування доповненої реальності у якому, на відміну від існуючих, використано нейронну мережу, що дозволяє урізноманітнити команди за рахунок взаємодії з віртуальними об’єктами в реальному світі.

Практична цінність отриманих результатів. Практична цінність одержаних результатів полягає в тому, що на основі отриманих в магістерській кваліфікаційній роботі теоретичних положень запропоновано алгоритми та розроблено програмний засіб для взаємодії користувач-комп’ютер на основі поєднання методів створення доповненої реальності і розпізнавання рухів для інтерактивних інформаційних систем.

Особистий внесок здобувача. У кваліфікаційній роботі автору належать наступні результати: метод класифікації з віднесенням зображення одразу до кількох класів, метод пришвидшеного додавання класу до нейромережі, програмне забезпечення для класифікації графічних зображень.

Апробація матеріалів магістерської кваліфікаційної роботи. Основні положення магістерської кваліфікаційної роботи доповідалися та обговорювалися на ІІ Всеукраїнській науково-практичній інтернет-конференції здобувачів вищої освіти і молодих учених "Інформаційно-комп'ютерні технології: стан, досягнення та перспективи розвитку" (м. Житомир, 2019) та ІІ Всеукраїнській науково-технічній конференції «Комп’ютерні технології: інновації, проблеми, рішення» (м. Житомир, 2019).

Публікації. Основні результати досліджень опубліковано в матеріалах двох конференцій.

1 АНАЛІЗ СУЧАСНОГО СТАНУ ВИРІШЕННЯ ЗАДАЧІ ТА ПОСТАНОВКА ЗАВДАННЯ

1.1 Аналіз систем розпізнавання зображень та рухів

Жестами можуть вважатися будь-які рухи або зміна положення тіла, але зазвичай вони означають рух особи або руки.

В ідеалі, розпізнавання жестів дозволить людині спілкуватися і взаємодіяти з машинами природно, без будь-яких механічних посередників. Використовуючи датчики, які виявляють рух тіла, технологія розпізнавання жестів дозволяє управляти пристроями, такими як телевізори, комп'ютери і відеоігри, в першу чергу за допомогою руху руки або пальця. За допомогою цієї технології користувач можете перемикати телевізійні канали, регулювати гучність і так далі.

Розпізнавання жестів дозволяє комп'ютерам бути більш доступними для людей з обмеженими фізичними можливостями та робить взаємодію більш природним в іграх або віртуальному світі 3D. Використовуючи розпізнавання жестів, можна вказувати пальцем на екран комп'ютера, а курсор буде відповідно переміщатися. Потенційно це може зробити непотрібними такі пристрої, як миші, клавіатури і навіть сенсорні екрани.

Системи, які можуть розпізнавати жести, покладаються на алгоритми. Розрізняють два різних алгоритмічних підходу в розпізнаванні жестів: на основі 3D і на основі зовнішнього вигляду. Найпопулярніший метод використовує 3D інформацію від основних частин тіла з метою отримання кількох важливих параметрів, таких як положення долоні або кут суглобів. На відміну від цього, системи на основі зовнішнього вигляду використовують для розпізнавання зображення або відео.

На додаток до технічних проблем реалізації технології розпізнавання жестів існують також соціальні проблеми. Жести повинні бути простими, інтуїтивними і універсально прийнятними. Для того щоб розпізнати об’єкти та

рухи не можливо обійтися без машинного навчання. Машинне навчання (ML) – це наукове навчання алгоритмів та статистичних моделей, які комп'ютерні системи використовують для виконання конкретного завдання без використання чітких інструкцій, спираючись на закономірності та умовиводи[3]. При традиційному комп'ютерному програмуванні програміст визначає правила, якими повинен користуватися комп'ютер. Однак ML потребує іншого мислення. В реальному світі ML зосереджується набагато більше на аналізі даних, ніж на кодуванні. Програмісти наводять набір прикладів, і комп'ютер вивчає шаблони з даних.

На сьогоднішній день існують два найпопулярніші фреймворки що дозволяють працювати з моделями нейронних мереж, а саме: TensorFlow та CoreML. В даному випадку модель – це результат застосування алгоритмів машинного навчання до набору навчальних даних.

TensorFlow – відкрита програмна бібліотека для машинного навчання, розроблена компанією Google для вирішення завдань побудови і тренування нейронної мережі з метою автоматичного знаходження та класифікації образів, досягаючи якості людського сприйняття [4].

CoreML – програмна бібліотека розроблена компанією Apple для інтеграції моделей машинного навчання в мобільні додатки та комп’ютери[5].

Все більшої популярності набирає система доповненої реальності. Її використовують в додатка, іграх, соціальних мережах. Доповнена реальність – термін, що позначає всі проекти, спрямовані на доповнення реальності будь-якими віртуальними елементами. Користувач за допомогою камери в його смартфоні може підібрати меблі в оселю, або одяг чи підняти собі настрій за допомогою різних масок, які накладаються на лице. Все це стало можливо реалізувати завдяки бібліотеці ARKit[6], яка й досі займає перше місце по відгукам користувачів.

Отже, поєднання даних технологій дає змогу створити систему розпізнавання рухів, а саме розпізнавання рухів людської долоні.

1.2 Порівняльний аналіз аналогів

На даний момент існує декілька реалізації мобільних додатків з доповненої реальністю та системою розпізнаванням рухів, нижче вказані деякі з них.

ARCore Elements (рисунок 1.1) – мобільний додаток створений компанією Google для демонстрації можливостей бібліотеки ARCore для пристроїв на базі операційної системи Android[7]. Додаток дозволяє розставляти елементи доповненої реальності на реальних поверхнях, та взаємодіяти з ними за допомогою екрану смартфону (рисунок 1.2).

Рисунок 1.1 – Демонстрація роботи додатку ARCore Elements

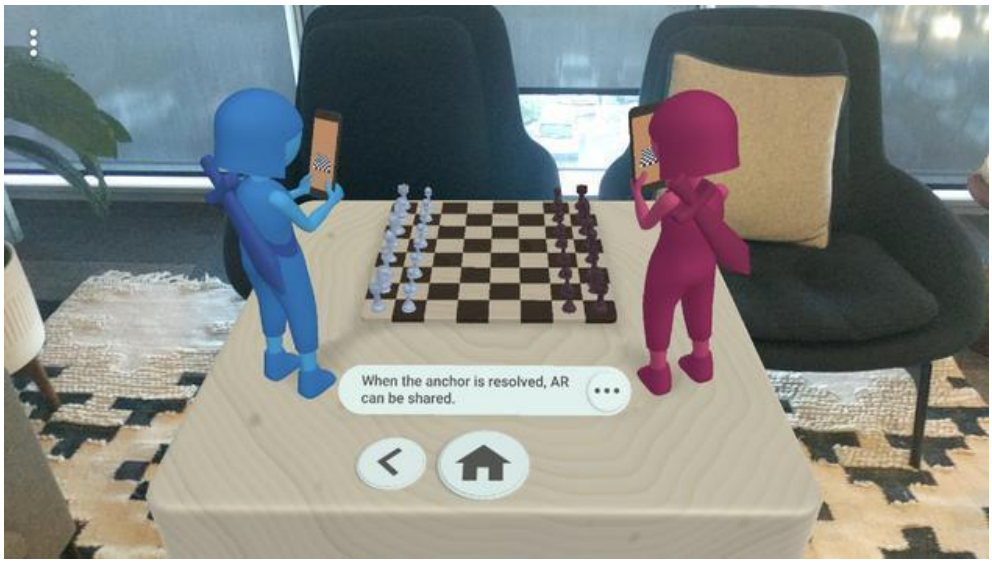


Рисунок 1.2 – Приклад взаємодії з об’єктами в додатку ARCore Elements

VR Gesture Player (рисунок 1.3) – мобільний додаток створений компанією Macroni для пристроїв на базі операційної системи Android[8]. Додаток дозволяє програвати відео доповненої реальності на керувати плеєром за допомогою рухів руки на вказівного пальця, які розпізнаються завдяки основній камері смартфону (рисунок 1.4).

Рисунок 1.3 – Керування плеєром VR Gesture Player(вид збоку)

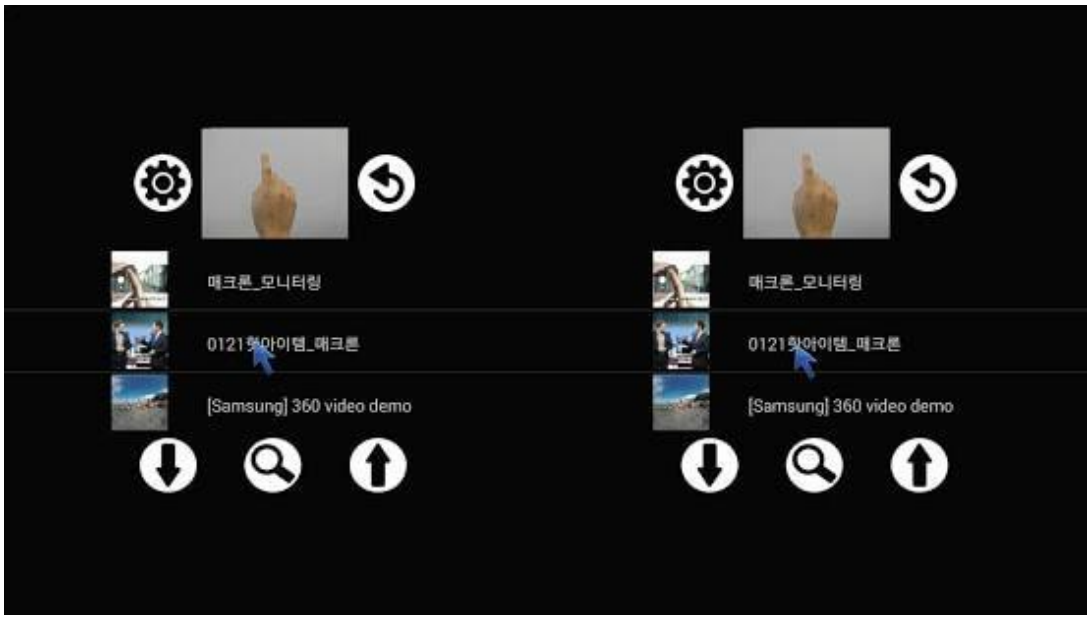


Рисунок 1.4 – Керування плеєром VR Gesture Player(вид в додатку)

Wanna Nails (рисунок 1.5) – мобільний додаток створений компанією WANNABY для пристроїв на базі операційної системи iOS[9]. Додаток дозволяє переглянути як виглядатиме лак для нігтів на руці користувача та замовити відповідний лак.

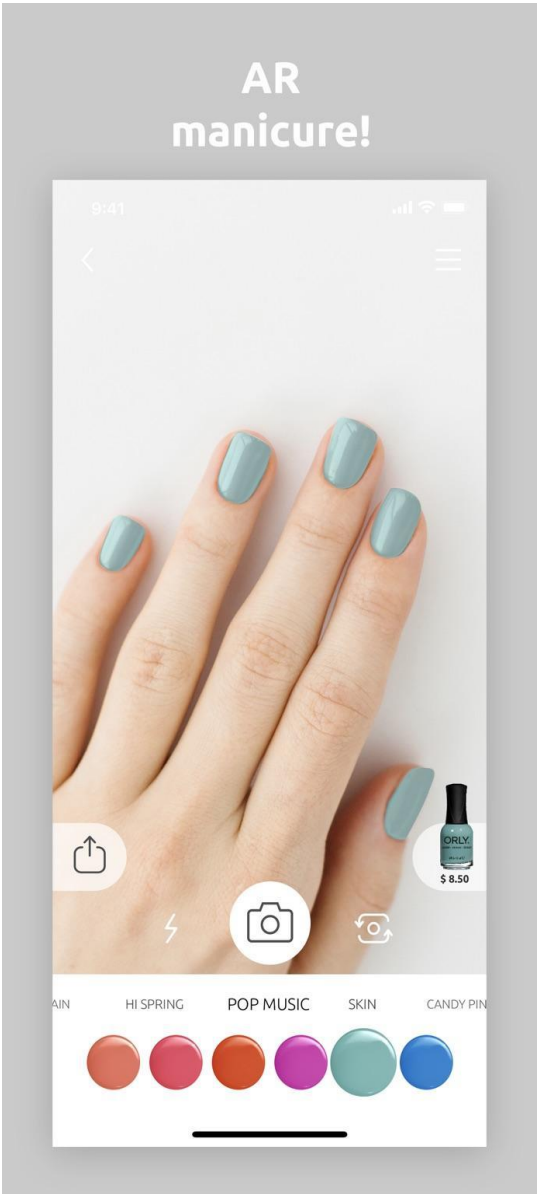


Рисунок 1.5 – Головне вікно програми Wanna Nails

Face Racer – no hands! (рисунок 1.6) – мобільний додаток створений компанією Sea Dragon Travels Pty Ltd для пристроїв на базі операційної системи iOS[10]. Мобільна гра, в якій потрібно повертати автомобілем за допомогою

нахилів голови вліво та вправо, а для збільшення швидкості автомобіля – необхідно посміхатись.

Рисунок 1.6 – Приклад роботи гри Face Racer – no hands!

Проаналізувавши усі аналоги, визначено їхні можливості та недоліки, які враховувались при створенні власного мобільного додатку з назвою «Hands Gesture AR» (табл. 1.1).

Таблиця 1.1 – Порівняльні характеристики програмних продуктів

Критерій	ARCore Elements	VR Gesture Player	Wanna Nails	Face Racer – no hands!	Hands Gesture AR
Розпізнавання рухів	-	+	+	+	+
Використання AR	+	-	+	-	+
Взаємодія з віртуальними об’єктами	+	+	-	+	+

Таблиця порівняльних характеристик показала, що розробка програмного продукту є актуальною та доцільною. В результаті отримаємо продукт, в якому відсутні недоліки існуючих рішень.

1.3 Постановка задачі розробки

Після аналізу стану програм тренувань та порівняння існуючих аналогів, було визначено наступні завдання, які необхідно виконати для розробки програмного продукту:

- розробити методи інтерактивної взаємодії з урахуванням розпізнавання жестів рук, положення долоні, кутів суглобів тощо;
- методи поєднання доповненої реальності і розпізнавання рухів;
- розробити модуль для роботи за доповненою реальністю;
- провести тестування програмного продукту.

1.4 Розробка технічних умов програмної системи

В процесі розробки було поставлено задачу, яку необхідно вирішити. Для цього потрібно проаналізувати переваги та недоліки існуючих систем і в результаті розробити власну систему, яка буде позбавлена недоліків що є у аналогів, при цьому не втрачаючи переваг.

Не зважаючи на те, що кожна з розглянутих програм використовує необхідний стек технологій для розпізнавання рухів та доповненої реальності – жодна з них не вирішує поставлену задачу.

Тому було прийнято рішення, створити власну систему, та доповнити її необхідними модулями для вирішення поставленої задачі. Для розробки планується використовувати нейронну мережу, яка буде розпізнавати рухи, елементи доповненої реальності, взаємодія з якими і є головною метою. Створені програмні модулі дозволять конвертувати розпізнаний об’єкт, а саме людську руку в пікселі та перевіряти зони зіткнення з віртуальним об’єктом.

Основні задачі які необхідно виконати:

- інтерфейс користувача;
- створення моделі для нейронної мережі;
- навчання моделі нейронної мережі;
- розробка модуля доповненої реальності;
- розробка модуля взаємодії користувача з віртуальними об’єктами.

Отже в результаті було визначено основні задачі, які необхідно виконати для вирішення проблеми, а саме – для взаємодії користувачі з світом доповненої реальності без використання спеціальних фізичних пристроїв.

1.5 Висновки

В даному розділі було розглянуто важливість розпізнавання рухів на сьогоднішній день та значимість використання смартфону в даних цілях. Також було проаналізовано стан вирішення даного питання шляхом розгляду аналогів, їх порівняння між собою та розроблюваною програмою. В результаті порівняння було відображено доцільність розробки та розроблено основні завдання, які необхідні для розробки програмного продукту.

2 РОЗРОБКА МЕТОДУ ІНТЕРАКТИВНОЇ ВЗАЄМОДІЇ КОРИСТУВАЧ-КОМП'ЮТЕР

2.1 Розробка методу поєднання доповненої реальності і розпізнавання рухів

На відміну від віртуальної реальності, яка дозволяє користувачу зануритись у віртуальне середовище, доповнена реальність – це спосіб розширення реальності за допомогою сучасних технологій.

Для роботи з елементами доповненої реальності було обрано фреймворк від Apple під назвою ARKit[11]. Для сцени необхідно створити ARViewController, який буде підписаний на делегати ARSessionDelegate, ARSCNViewDelegate, що дозволить отримувати дані з камери смартфону для подальшої обробки. Головним view елементом виступає об'єкт ARSCNView. Для розширення реальності віртуальними об'єктами їх необхідно додати в головний view елемент. ARKit автоматично розпізнає площину за допомогою класу конфігурації ARWorldTrackingConfiguration. Для того щоб об'єкти на площині підпорядковувались фізичним законам, кожного разу при оновленні площини необхідно оновлювати SCNGeometry, тому було прийнято рішення створити власний клас для роботи з площиною в якому будуть реалізовані всі необхідні методи.

2.2 Розробка методу інтерактивної взаємодії з урахуванням розпізнавання жестів рук, положення долоні, кутів суглобів тощо

Сучасний рівень розвитку інформаційних технологій, алгоритмів і методів сприяє появі нових інтерфейсів взаємодії користувачів і пристроїв. Одним з перспективних варіантів є розробка і дослідження людино-машинних інтерфейсів, заснованих на розпізнаванні об'єктів. Перед розробниками подібних інтерфейсів ставиться завдання використання природних для людини способів

спілкування з комп'ютерами за допомогою жестів, голосу, міміки та інших модальностей.

Одним з основних завдань розширення взаємодії людини з комп'ютером є розробка загальної методології виявлення і розпізнавання динамічних жестів людини, що здійснюються руками. Для вирішення цього завдання необхідно на основі аналізу існуючих методів виявлення і розпізнавання жестів людини, розробити алгоритм аналізу потокового відео і розпізнавання жестів. В даний час існує безліч різних методів виявлення об'єкта на зображенні. Ці методи можна розділити на три основні групи[12]:

- скелетні методи;
- методи на основі 3D моделі об'єкта;
- методи на основі 2D моделі об'єкта.

У скелетних методах досліджується контур силуету: зазвичай відшукуються кути, виступи, западини і інші точки з високими значеннями кривизни. Для отримання інформації про форму контуру застосовуються різні уявлення кордону об'єкта. У методах на основі 3D моделі руки представляють у вигляді складних тривимірних поверхонь і класифікуються за допомогою нейронних мереж. Метод 2D розпізнавання схожий з попереднім, але використовує двовимірне зображення замість об'ємних моделей.

Кожен з методів має переваги і недоліки. Контурний аналіз – метод опису та пошуку графічних образів з їх контурам. В даному контексті контур – це крива, яка описує межі об'єкта на зображенні. Передбачається, що дана лінія містить достатньо інформації про форму об'єкта, при цьому його внутрішні точки не враховуються. Розгляд тільки контурів об'єктів дозволяє піти від простору зображення до простору контурів, що знижує складність алгоритмів і обчислень. Оскільки при розробці алгоритму об'єктом виявлення на зображенні є рука, то перший метод можна виключити через незручність використання контуру для визначення конкретного жесту. Недоліком методу на основі 3D моделі є його ресурсомісткість. Побудова 3D моделі, навчання нейронної мережі

та її використання можуть вимагати значних ресурсів, так само не варто забувати, що для використання даного методу вимагає камери з можливістю визначення глибини зображення. Тому найчастіше використовуються 2D методи, так як в них знижується обчислювальна складність і відпадає необхідність в спеціальному обладнанні, оскільки для отримання зображення може бути використана звичайна веб-камера. Оскільки 2D методи є менш точними, виникає необхідність в їх удосконаленні для отримання не ресурсоємкого і ефективного алгоритму розпізнавання об'єктів.

Одним з перспективних методів розпізнавання образів вважається метод Віоли-Джонса – алгоритм, що дозволяє виявляти об'єкти на зображеннях в реальному часі. Цей метод запропонований в 2001 році Полом Віолою і Майклом Джонсом. Даний метод є основним для пошуку об'єктів на зображенні в реальному часі в більшості існуючих алгоритмів розпізнавання та ідентифікації. Так само він є одним з кращих по співвідношенню ефективності розпізнавання і швидкості роботи. Алгоритм показує відмінні результати і розпізнає об'єкти під невеликим кутом, приблизно до 30 градусів, і при різних умовах освітленості.

Основні принципи, на яких базується метод:

1. Можливість подання зображення в інтегральному вигляді, що дозволяє обчислювати швидко необхідні об'єкти;

2. Використання ознак Хаара, тобто ознак цифрового зображення, які використовуються в розпізнаванні образів, за допомогою яких відбувається пошук потрібного об'єкта;

3. Застосування алгоритму поліпшення для вибору найбільш підходящих ознак для шуканого об'єкта на даній частині зображення, процедури послідовної побудови композиції алгоритмів машинного навчання, коли кожен наступний алгоритм прагне компенсувати недоліки композиції всіх попередніх алгоритмів;

4. Використання каскадів ознак для швидкого відкидання вікон, де не знайдений об'єкт.

Метод Віоли-Джонса з використанням ознак Хаара є одним з кращих алгоритмів для вирішення задач виявлення об'єктів на зображеннях в реальному часі, використовуючи двомірні зображення. Проте його використання вимагає значних системних ресурсів і тому не є оптимальним рішенням.

Отже, було розглянуто основні методи знаходження і розпізнавання об’єктів на відео в режимі реального часу. Було описано переваги та недоліки кожного з цих методів, проте найкращим методом виявився метод Віоли-Джонса з використанням ознак Хаара. Реалізація даного методу потребує значних обчислень та перетворень для отримання бажаного результату.

Саме тому, було прийнято рішення використовувати метод для розпізнавання і знаходження рухів руки, реалізація якого потребує наявності нейронної мережі, що позбавить програміста від зайвої кількості формул та покращить результати самого розпізнавання завдяки навчанню даної мережі. Існуючі методи розпізнавання рухів використовують для нейронної мережі 3D моделі, але цей тип вхідних даних потребує великої кількості ресурсів як для обробки, так і для збереження. Отже найбільш оптимальним варіантом буде використання 2D фотографій для навчання нейронної мережі. Для знаходження об’єкта на фотографії нейронній мережі необхідно мати фотографії людських рук та маски для цих фотографій, тобто зображення які мають чорний фон та білий силует людської руки (рисунок 2.1).

Рисунок 2.1 – Приклад маски зображення людської руки

Наступним кроком є взаємодія людської руки, яка була розпізнана нейронною мережею, з віртуальними об’єктами в реальному світі для тестування системи. Тому, було прийнято рішення створити метод, який буде знаходити найвищу точку розпізнаного об’єкта (рисунок 2.2). Для цього необхідно отримане зображення руки перевести в пікселі, пошук потрібної точки виконувати знизу – вгору.

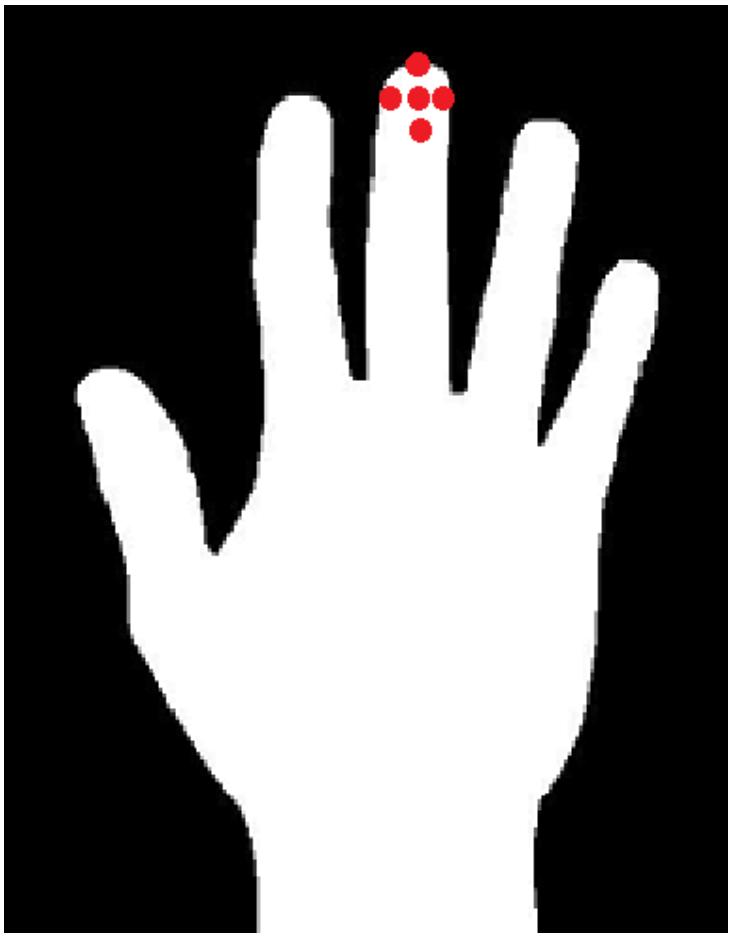


Рисунок 2.2 – Приклад розпізнавання найвищої точки об’єкта

Спочатку виконується перевірка чи піксель білий, якщо так – то перевіряються чотири сусідніх пікселя. Даний алгоритм зображено на рисунку 2.3.

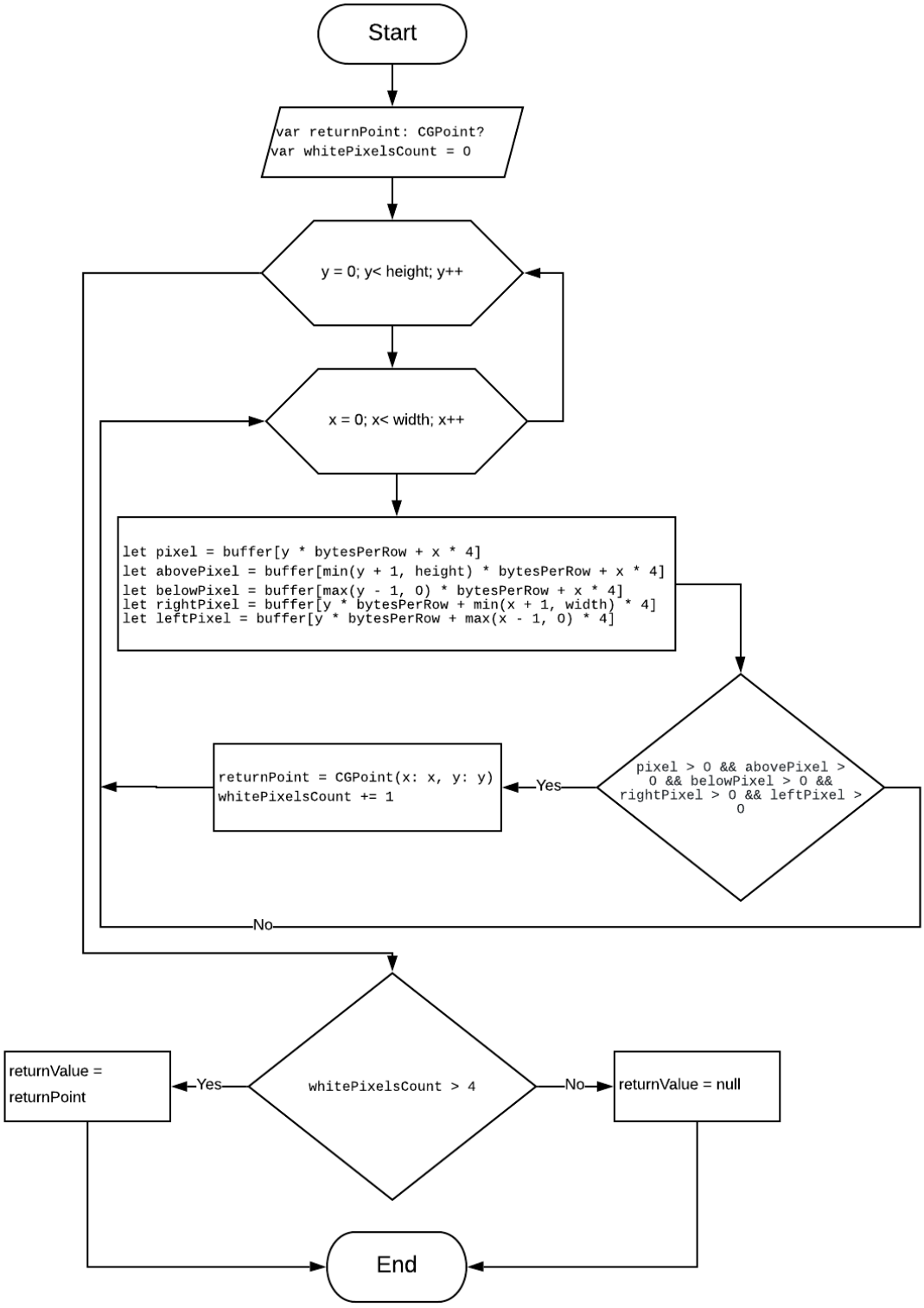


Рисунок 2.3 – Алгоритм знаходження найвищої точки

В результаті отримуємо точку в центрі якої, для тестування, можна помістити віртуальний об’єкт, якщо точка дорівнює null – це означає що не дані людську руку було розпізнано з помилками.

2.3 Навчання моделі семантичної сегментації на основі нейронної мережі

Знаходження людської руки на відео в реальному часу є досить популярної проблемою, проте в сучасному світі нейронні мережі набирають все більшої популярності. Тому було прийнято рішення використовувати нейронні мережі для розпізнавання рухів людської руки.

Вихідним пунктом є вибір алгоритму. Було розглянуто можливості, які пропонують Create ML та Turi Create[13]. Create ML, не має функціоналу для виявлення та відстеження об'єктів, однак це можливо Turi Create, але на виході моделі буде обмежувальне поле (рисунок 2.4).

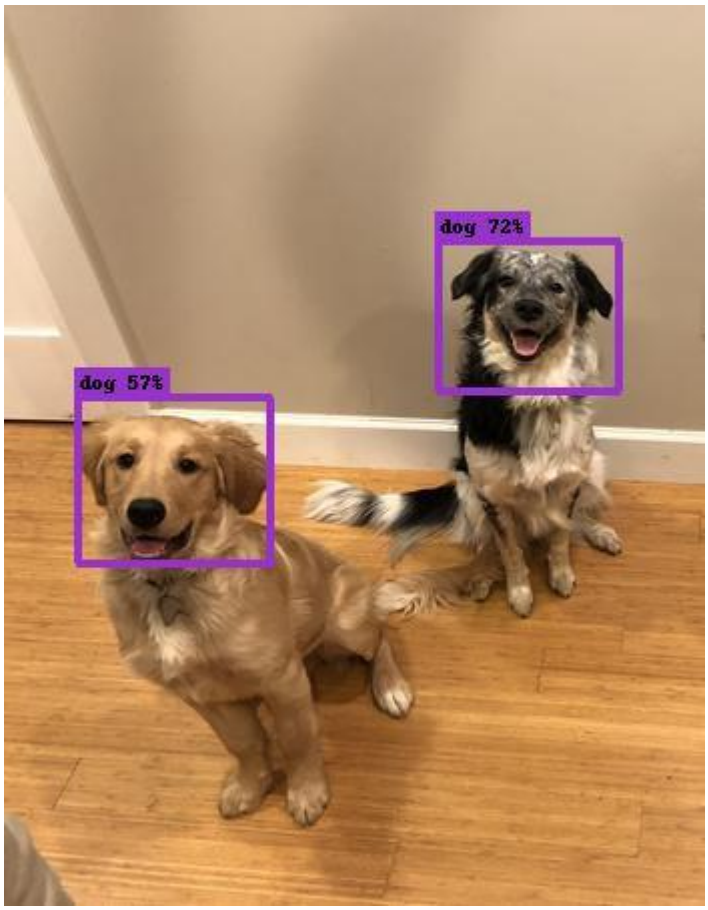


Рисунок 2.4 – Приклад виявлення об’єктів з Turi Create

Розглянуті рішення не підходять для вирішення поставленої задачі. Отже, в результаті дослідження було прийнято рішення розпізнавати об’єкти на рівні пікселів за допомогою необхідних методів та класів які були описані в пункті 2.2.

Для нейронної мережі необхідно обрати розмір сітки. Оптимальним розміром з точки зору точності, швидкодії та ресурсів є 224x224. Для навчання моделі було обрано технології PyTorch[14], редактор Jupyter[15] та інші додаткові бібліотеки [рисунок 2.5].

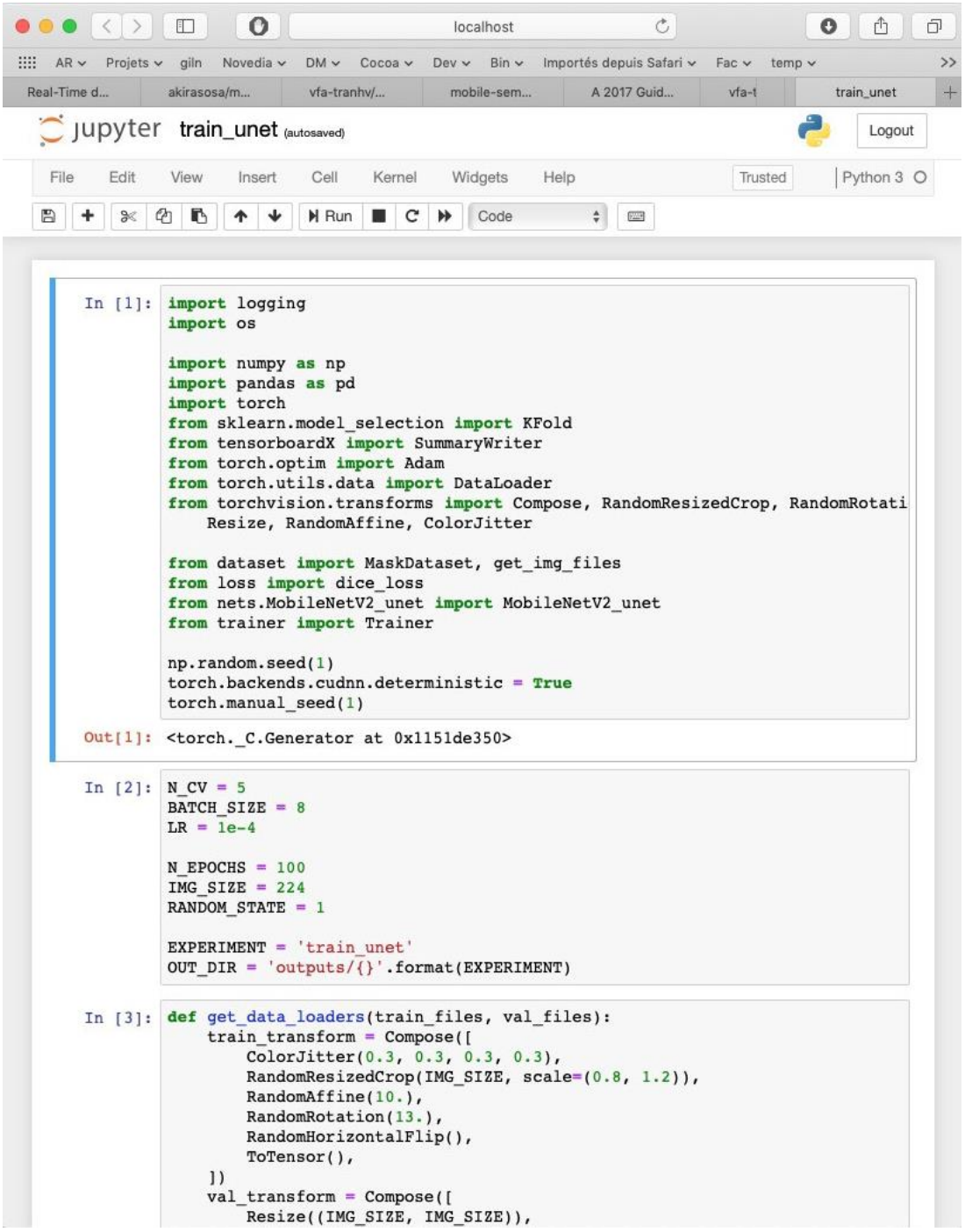


Рисунок 2.5 – Навчання моделі нейронної мережі

Згідно документації для початку навчання потрібно щонайменше 30 прикладів об’єктів, але для кращої якості кількість прикладів має бути більше 200. Результати навчання можна побачити на рисунку 2.6.

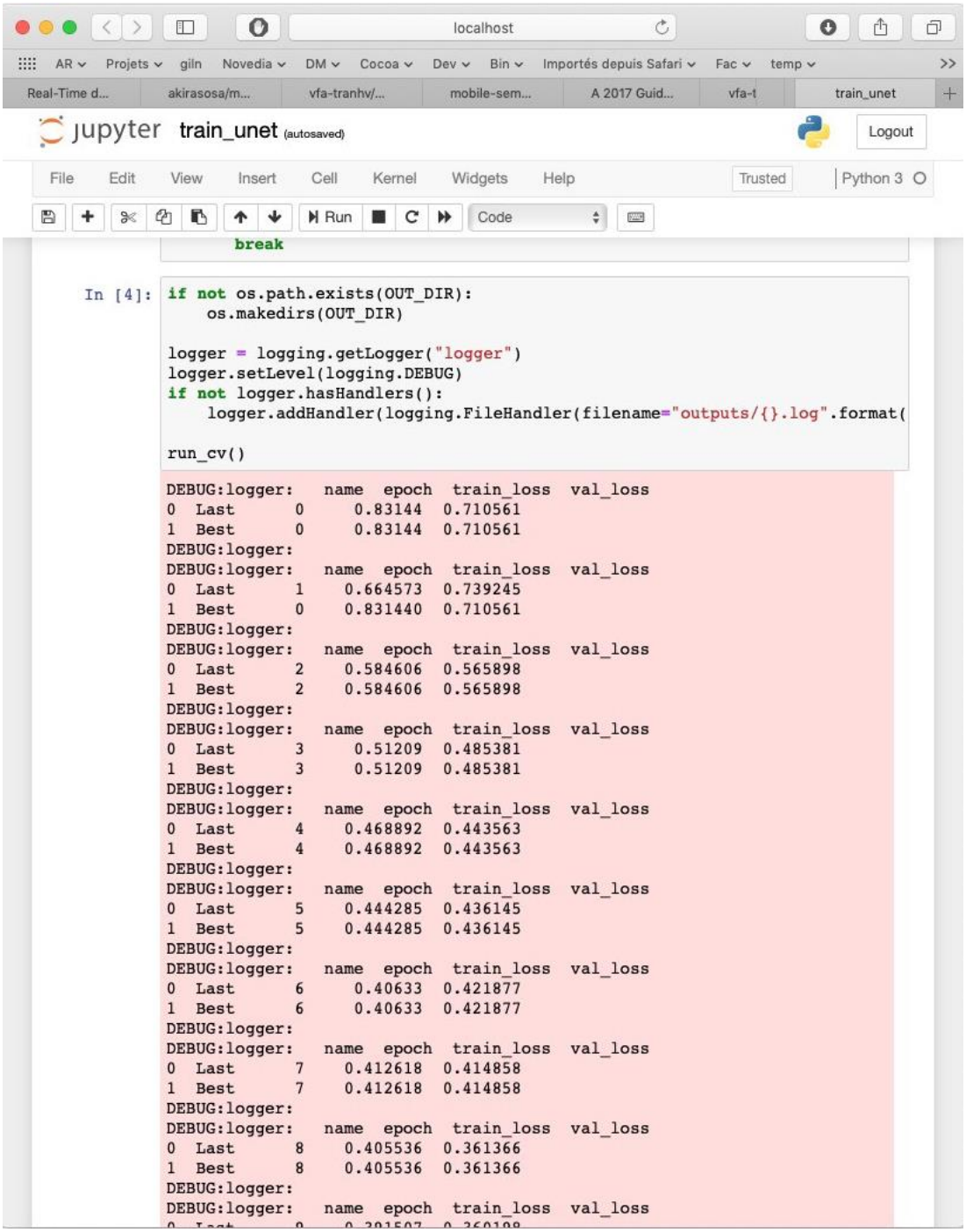


Рисунок 2.6 – Результат навчання моделі нейронної мережі

В результаті навчання була створена необхідна модель. Команда python coreml_covnerter.py – конвертує модель в необхідний формат для бібліотеки CoreML.

2.4 Розробка інтерфейсу програмного додатку

Інтерфейс користувача – це точка взаємодії людини та комп'ютера в пристрої. Сюди можна віднести екрани дисплея, клавіатури, мишу та зовнішній вигляд робочого столу. Це також спосіб взаємодії користувача з додатком чи веб-сайтом. Зростаюча залежність багатьох підприємств від веб-додатків та мобільних додатків призвела до того, що багато компаній надають підвищений пріоритет користувальницьким інтерфейсам, намагаючись покращити загальний досвід користувача. Проте, для системи розпізнавання рухів інтерфейс має бути простим і зрозумілим. Саме тому було прийнято рішення створити його з максимально малої кількістю елементів (рисунок 2.7).

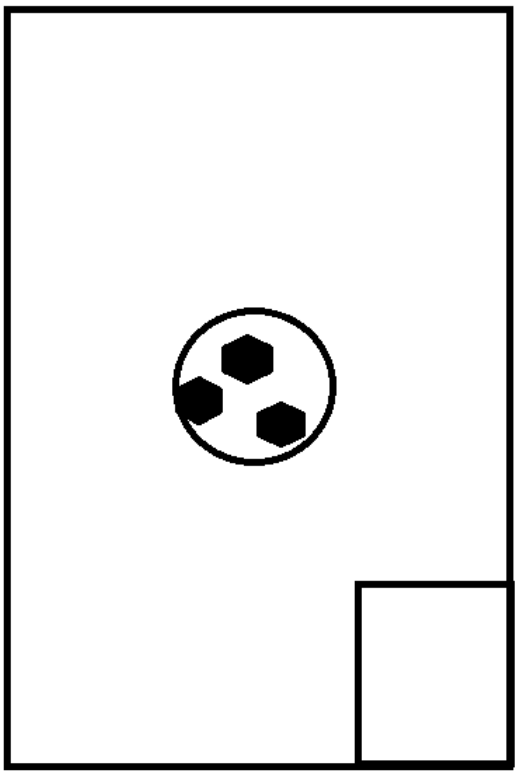


Рисунок 2.7 – Інтерфейс додатку Hands Gesture AR

В нижньому правому кутку знаходиться відображення результату розпізнавання людської руки. Для тестування додатку, в центр простору доповненої реальності було прийнято рішення помістити віртуальний об’єкт – м’яч. Таким чином, користувач зможе за допомогою смартфону та вбудованої камери створити віртуальний об’єкт та взаємодіяти з ним.

2.5 Висновки

В даному розділі було обрано фінальний тип інтерфейсу. Розглянуто методи розпізнавання рухів, в результаті чого було прийнято рішення створити власний метод який поєднує в собі всі переваги. Також було навчено модель нейронної мережі на основі фотографій та масок для них.

3 РОЗРОБКА МОБІЛЬНОГО ДОДАТКУ

3.1 Варіантний аналіз і обґрунтування вибору засобів реалізації програмного продукту.

Apple iOS – це мобільна операційна система, яка не потребує представлення. Ця мобільна платформа використовується на мільйонах пристроїв iPhone, iPad та iPod і створює необмежену кількість можливостей для створення високоякісних інноваційних програм. Однак розробити якісний додаток для iOS неможливо без правильного набору мов та інструментів. До найпопулярніших мов програмування, які використовуються для розробки програм для iOS можна віднести: Objective-C, Swift, C#, Python, C++, HTML 5.

Objective-C був розроблений Томом Лав і Бредом Коксом у 1984 році[16]. До запуску Apple Swift в 2014 році, Objective C була основною мовою мобільних додатків Apple iOS. Objective-C – об'єктно-орієнтована мова програмування. Передача повідомлень між об'єктами – ключова особливість Objective-C, яка стала дійсно корисною для операційних систем Apple iOS. На сьогоднішній день Swift випередив Objective-C за популярністю та корисністю. Objective-C – це поєднання мови програмування на C і об'єктно-орієнтованих можливостей. Objective-C успадковує синтаксис, примітивні типи та оператори управління потоком C та додає синтаксис для визначення класів та методів. Він також додає підтримку на мовному рівні для управління об'єктними графами та літералами об'єктів, забезпечуючи при цьому динамічне введення та прив'язування, відкладаючи багато обов'язків до виконання.

Swift – це основна мова програмування операційної системи iOS. Swift була розроблена та запущена Apple у 2014 році[17]. У грудні 2015 року компанія Apple відкрила Swift під ліцензією Apache License 2.0. Окрім iOS, Swift також є мовою програмування для macOS, watchOS, tvOS, Linux та z/OS. До початку Swift, Objective-C була основною мовою для розробки iOS. Objective-C, мова яка існує понад 30 років, не підтримувала сучасних потреб. Swift – це сучасна мова

програмування, яка забезпечує такі сучасні мовні функції, як динамічне, безпечне прив'язування та розширюваність. На початку 2018 року Swift випередив за популярністю Objective-C і став мовою програмування №1 для iOS та інших операційних систем Apple. Swift – це рекомендована мова для створення нових додатків на iOS, tvOS та watchOS. Розробка iOS зі Swift 4 включає ARKit, CoreML та багато іншого.

Мова C# була створена Андерсом Хейлсбергом в Microsoft і була запущена в 2000 році[18]. C# – це проста, сучасна, гнучка, об'єктно-орієнтована, безпечна та відкрита мова програмування. C# – одна з найбільш універсальних мов програмування у світі, яка дозволяє розробникам створювати всі типи програм, включаючи клієнтів Windows, консолі, веб-додатки, мобільні додатки та додаткові системи. C# розробники можуть створювати вбудовані мобільні додатки для iOS та Android за допомогою Xamarin. Xamarin – це інструмент у складі Visual Studio, який дозволяє розробникам писати код C#, який компілюється в нативний код в iOS та вбудовані бінарні файли на Android. Ці бінарні файли працюють точно так само, як і будь-яка вбудована програма, написана на інших мовах iOS та Android, таких як Swift або Kotlin. C# пропонує розробникам можливість створювати рідні мобільні додатки для iOS та Android, не вивчаючи нової мови програмування.

Python – одна з найпопулярніших мов програмування останніх часів. Python, створений Гідо ван Россумом у 1991 році, є програмою з відкритим кодом, на високому рівні, програмою загального призначення[19]. Python – це динамічна мова програмування, яка підтримує об'єктно-орієнтовані, імперативні, функціональні та процедурні парадигми розвитку. Python дуже популярний у програмуванні машинного навчання. У розробці додатків для iOS Python можна використовувати для створення бібліотек, функцій та завдань із зворотної обробки.

C++ – одна з найдавніших і найпопулярніших мов програмування[20]. У розробці Android C++ використовується для створення API-програм та методів

для доповнення якогось функціоналу. Є кілька популярних вбудованих бібліотек C++, які розробники iOS можуть використовувати у своїх додатках.

HTML 5 у поєднанні з CSS та іншими технологіями можна використовувати для створення гібридних додатків iOS. Ці програми не є нативними програмами. Гібридні додатки працюють на базі браузера та використовують HTML та CSS для візуалізації сторінок у програмах.

Не менш важливим є вибір середовища розробки. XCode – офіційний IDE Apple для розробки програм для всіх платформ Apple. Включаючи все необхідне для створення якісних додатків для iOS, він слугує редактором вихідного коду, виступає інструментом відлагодження та виявлення і усунення неполадок із коду додатка. Будучи надзвичайно швидким і плавним, це перший вибір розробників для створення додатків для iOS.

Розроблений JetBrains, AppCode – це інтелектуальне IDE для розробки додатків для iOS/macOS. Завдяки глибокому розумінню структури коду, він обробляє звичайні завдання, регулярно контролюючи якість коду, і позбавляє від додаткового набору тексту. Підтримує ряд мов програмування, включаючи Swift, Objective-C, C ++, JavaScript тощо, AppCode полегшує розробку додатків для iPhone, Mac, iPad, Apple Watch та Apple TV.

Отже, на основі аналізу мов програмування було прийнято рішення обрати мову програмування Swift, так як вона включає в собі всі необхідні елементи для вирішення поставленої задачі. Середовищем розробки обрано Xcode, який рекомендований Apple, та містить безліч інструментів для створення якісних продуктів.

3.2 Розробка модуля знаходження людської руки

Для вирішення поставленої задачі необхідно створити клас, який дозволить працювати з елементами доповненої реальності. В iOS розробці за це відповідає клас ARSCNView. Отже потрібно створити UIViewController та додати до нього екземпляр класу ARSCNView як базовий view елемент. Нижче

наведено код головного екрану з базовим view елементом для роботи з доповненою реальністю.

```
class ARViewController: UIViewController {
   let sceneView = ARSCNView()

   override func loadView() {
      super.loadView()

      view = sceneView
      let configuration = ARWorldTrackingConfiguration()
      sceneView.session.run(configuration)
   }
}
```

ARSCNView – це елемент, який поєднує SCNScene від SceneKit з ARSession від ARKit. SceneKit – це 3D-бібліотека від Apple, яка дозволяє завантажувати та відображати 3D-об’єкти. Це повноцінний 3D-двигун з безліччю функцій, таких як фізика, освітлення тощо. ARWorldTrackingConfiguration було додано для перегляду каналу камери. Метод func session(_ session: ARSession, didUpdate frame: ARFrame) дозволяє аналізувати поточний кадр за допомогою CoreML. Властивість captureImage з ARFrame є CVPixelBuffer, який ідеально підходить для запитів CoreML. Не слід забувати, що для вирішення поставленої задачі необхідно аналізувати відео в реальному часі. Делегат викликається 30 разів на секунду. Якщо аналіз кадру займатиме занадто довго, то відео почне пропускати кадри, і воно почне виглядати заторможеним. Щоб цього не допустити необхідно додати перевірку яка дозволить обробляти кадр, при цьому не впливаючи на роботу відео. В результаті було створено змінну currentBuffer: CVPixelBuffer? та додано

наступну перевірку.

```
guard currentBuffer == nil, case .normal = frame.camera.trackingState else {
        return
}
currentBuffer = frame.capturedImage
startDetection()
```

В методі startDetection змінна currentBuffer зануляється та відбувається запит до CoreML. Для використання нейронної моделі необхідно підключити дві бібліотеки CoreML та Vision. Запит для знаходження людської руки створюється завдяки змінній, код якої наведено нижче.

```
var MLRequest: VNCoreMLRequest = {
   do {
      let model = try VNCoreMLModel(for: HandModel().model)
      let request = VNCoreMLRequest(model: model)
request.imageCropAndScaleOption = VNImageCropAndScaleOption.scaleFill
      return request
   } catch {
      fatalError("can't load Vision ML model: \(error)")
   }
}()
```

За виконання даного запиту відповідає функція StartDetection(). Орієнтацію камери під час обробки запиту необхідно вказати right, так як камера iOS пристроїв перевартає її в положення landscape.

```
func startDetection() {
 ...
```

```
    let requestHandler = VNImageRequestHandler(cvPixelBuffer: buffer, orientation:
.right)
    bgQueue.async {
        ...
    }
}
```

Результати виконання запиту доступні в observation.pixelBuffer. Отримані дані буде відображено в спеціальному view елементі, що знаходиться внизу екрану (рисунок 3.1).

Рисунок 3.1 – Результат розпізнавання людської руки

Наступним кроком є робота з SceneKit, щоб додати віртуальні об'єкти у тривимірний простір. Потрібно використовувати ARKit, щоб прив’язати цей віртуальний 3D-простір у реальному світі, який користувач бачить через камеру iPhone. Це дасть ілюзію, що віртуальний об’єкт справжній. За допомогою ARKit камера може виявляти точки, які є характерними для певної області зображення камери. Згідно документації,позиції цих точок в просторі координат 3D світу екстраполюються як частина аналізу зображень, який проводить ARKit, щоб точно відслідковувати положення, орієнтацію та рух пристрою. У сукупності ці точки співвідносяться з контурами реальних об'єктів з каналу камери.

Після початку виявлення площин в ARKit, він проаналізує ці особливості, і якщо деякі з них є компланарними, система використовуватиме їх для оцінки форми та положення поверхні.

У коді використовується ARSCNView, який може автоматично визначати точки функцій (ARAnchor) та площини (ARPlaneAnchor). Щоб увімкнути виявлення площини, потрібно налаштувати ARSession.

```
...
        let configuration = ARWorldTrackingConfiguration()
        configuration.planeDetection = .horizontal

        sceneView.session.delegate = self

        sceneView.session.run(configuration)
…
```

Тепер необхідно задовільнити вимоги ARSCNViewDelegate протоколу. Для цього необхідно додати метод func renderer(_ renderer: SCNSceneRenderer, didAdd node: SCNNode, for anchor: ARAnchor).

SCNNode використовується SceneKit як елемент графу сцени. Об'єкт вузла сам по собі не має видимого вмісту. Він представляє лише координатне перетворення простору (положення, орієнтація, масштаб) відносно його батьківського.

Сцена ARSCNView має rootNode і автоматично заповнює простір SceneKit реального світу.

Оскільки активовано виявлення площин у ARKit, тепер є можливість створити віртуальні площини, які відповідають виявленим площинам. Таким чином, це дасть ілюзію, що віртуальні об’єкти фактично розміщені на об’єктах реального світу. Для цього було створено клас PlaneNode.

```
class PlaneNode: SCNNode {

    public func update(from planeAnchor: ARPlaneAnchor) {
        guard let device = MTLCreateSystemDefaultDevice(),
          let geom = ARSCNPlaneGeometry(device: device) else {
            fatalError()
        }

        geom.firstMaterial?.diffuse.contents =
UIColor.blue.withAlphaComponent(0.3)
        geom.update(from: planeAnchor.geometry)
        geometry = geom
    }
```

PlaneNode має спеціальний тип геометрії, доступний в ARKit 1.5 — ARSCNPlaneGeometry. Ця геометрія створюється ARKit, коли вона виявляє площину, щоб форма була максимально наближена до реальної площини. Таким чином ми можемо мати наприклад круглі столи.

Потрібно оновлювати цю геометрію щоразу, коли ARKit виявляє зміну геометрії виявленої площини.

```
func renderer(_: SCNSceneRenderer, nodeFor anchor: ARAnchor) -> SCNNode? {
    guard let _ = anchor as? ARPlaneAnchor else { return nil }

    return PlaneNode()
}

func renderer(_: SCNSceneRenderer, didAdd node: SCNNode, for anchor: ARAnchor) {
    guard let planeAnchor = anchor as? ARPlaneAnchor,
        let planeNode = node as? PlaneNode else {
        return
    }
    planeNode.update(from: planeAnchor)
}

func renderer(_: SCNSceneRenderer, didUpdate node: SCNNode, for anchor: ARAnchor) {
    guard let planeAnchor = anchor as? ARPlaneAnchor,
        let planeNode = node as? PlaneNode else {
        return
    }
    planeNode.update(from: planeAnchor)
}
```

Тепер на створеній штучній площині потрібно розмістити віртуальний об’єкт.

```
public class BallNode: SCNNode {

    public convenience init(radius: CGFloat) {
        self.init()
        let sphere = SCNSphere(radius: radius)

        let reflectiveMaterial = SCNMaterial()
        reflectiveMaterial.lightingModel = .physicallyBased

        reflectiveMaterial.metalness.contents = 1.0

        reflectiveMaterial.roughness.contents = 0.0
        sphere.firstMaterial = reflectiveMaterial
        self.geometry = sphere

        physicsBody = SCNPhysicsBody(type: .dynamic, shape: nil)

        let obj = SCNScene(named: "3dModel.scnassets/ball.scn")!

        let wrapperNode = SCNNode(geometry: sphere)

        for child in obj.rootNode.childNodes {
            child.scale = SCNVector3Make(Float(radius), Float(radius), Float(radius))

            wrapperNode.addChildNode(child)
        }
```

```
        self.addChildNode(wrapperNode)
    }
}
```

В результаті було створено віртуальний об’єкт у вигляді футбольного м’яча. Для того щоб користувач мав змогу взаємодіяти з віртуальним об’єктом, цей об’єкт потрібно додати до головної сцени нашого додатку. Для цього створимо об’єкт UITapGestureRecognizer, який реагує на дотик до екрану, після його спрацьовує делегат в якому потрібно прописати код розміщення віртуального об’єкта на сцені. Лістинг, який вказано нижче – розміщує віртуальний об’єкт по натиску на екран. на сцені (рисунок 3.2).

```
        ...
        let tap = UITapGestureRecognizer(target: self, action:
#selector(self.handleTap(_:)))
        myView.addGestureRecognizer(tap)
        ...

@objc func handleTap(_ sender: UITapGestureRecognizer? = nil) {
                …

                let ball = BallNode(radius: 0.05)
                ball.simdTransform = hitTestResult.worldTransform
                ball.position.y += 0.05
                sceneView.scene.rootNode.addChildNode(ball)
                …
        }
        …
```

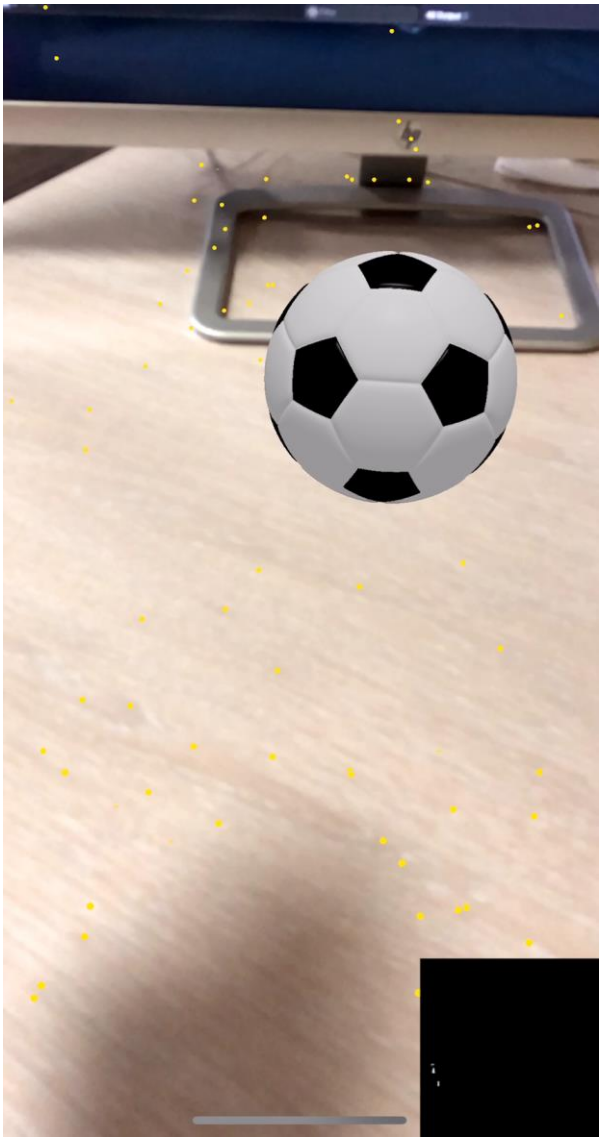


Рисунок 3.2 – Розміщення віртуального об’єкта на сцені

Тепер необхідно аби віртуальний м’яч підкорявся фізичним законам. Для цього до класу BallNode було додано: physicsBody = SCNPhysicsBody(type: .dynamic, shape: nil). І модифіковано функціх оновлення для PlaneNode.

```
func update(from planeAnchor: ARPlaneAnchor) {
      guard let device = MTLCreateSystemDefaultDevice(),
       let geom = ARSCNPlaneGeometry(device: device) else {
          fatalError()
    }
```

```
            geom.firstMaterial?.diffuse.contents =
UIColor.blue.withAlphaComponent(0.3)
            geom.update(from: planeAnchor.geometry)

             geometry = geom

            let shape = SCNPhysicsShape(geometry: geom, options:
[SCNPhysicsShape.Option.type: SCNPhysicsShape.ShapeType.boundingBox])

            physicsBody = SCNPhysicsBody(type: .static, shape: shape)
        }
```

В ході розробки було прийнято рішення для більшої реалістичності додати до віртуального об’єкта тінь. Для цього потрібно модифікувати planeNode.

```
        ...
        let material = SCNMaterial()
        material.lightingModel = .constant
        material.writesToDepthBuffer = true
        material.colorBufferWriteMask = []
        geom.firstMaterial = material

        geom.update(from: planeAnchor.geometry)
        …
```

Також необхідно створити новий клас SpotlightNode, який відповідає за освітлення.

```
        public class SpotlightNode: SCNNode {
```

```
        public override init() {
            super.init()
            commonInit()
        }

        required init?(coder aDecoder: NSCoder) {
            super.init(coder: aDecoder)
            commonInit()
        }

        private func commonInit() {
            let spotLight = SCNLight()

            spotLight.type = .directional
            spotLight.shadowMode = .deferred
            spotLight.castsShadow = true
            spotLight.shadowRadius = 100.0
            spotLight.shadowColor = UIColor(red: 0.0, green: 0.0, blue: 0.0, alpha:
0.2)

            light = spotLight

            eulerAngles = SCNVector3(-Float.pi / 2, 0, 0)
        }
    }
```

Методом спроб було обрано найкращу позицію для прожектора, а саме розмістити її на 10 метрів над користувачем. Це додає ще більшої реалістичності об’єкту.

3.3 Розробка модуля взаємодії людської руки з віртуальним об’єктом

Отже, повернемося до проблеми з виявленням рук. Результатом обробки зображень за допомогою CoreML є pixelBuffer, в якому чорні пікселі це фон, а білі пікселі – це розпізнані людські руки. Для конвертування в координати 3В світу буде використано алгоритм запропонований в другому розділі. Для цього потрібно створити розширення для CVPixelBuffer.

```
extension CVPixelBuffer {
    func searchTopPoint() -> CGPoint? {
        let width = CVPixelBufferGetWidth(self)
        let height = CVPixelBufferGetHeight(self)

        let bytesPerRow = CVPixelBufferGetBytesPerRow(self)

        CVPixelBufferLockBaseAddress(self,
CVPixelBufferLockFlags(rawValue: 0))

        defer {
            CVPixelBufferUnlockBaseAddress(self,
CVPixelBufferLockFlags(rawValue: 0))
        }

        var returnPoint: CGPoint?

        var whitePixelsCount = 0
```

```
            if let baseAddress = CVPixelBufferGetBaseAddress(self) {
                let buffer = baseAddress.assumingMemoryBound(to: UInt8.self)

                for y in (0 ..< height).reversed() {
                    for x in (0 ..< width).reversed() {

                        let pixel = buffer[y * bytesPerRow + x * 4]
                        let abovePixel = buffer[min(y + 1, height) * bytesPerRow + x * 4]
                        let belowPixel = buffer[max(y – 1, 0) * bytesPerRow + x * 4]
                        let rightPixel = buffer[y * bytesPerRow + min(x + 1, width) * 4]
                        let leftPixel = buffer[y * bytesPerRow + max(x – 1, 0) * 4]

                        if pixel > 0 && abovePixel > 0 && belowPixel > 0 && rightPixel
> 0 && leftPixel > 0 {
                            let newPoint = CGPoint(x: x, y: y)
                            returnPoint = CGPoint(x: newPoint.x / CGFloat(width), y:
newPoint.y / CGFloat(height))
                            whitePixelsCount += 1
                        }
                    }
                }
            }

            if whitePixelsCount < 10 {
                returnPoint = nil
            }
            return returnPoint
        }
```

```
}
```

В результаті виконання даного алгоритму буде отримано найвищу точку розпізнаної руки. Для взаємодії руки з м’ячем необхідно помістити в знайдену точку невидимий віртуальний об’єкт. Для цього було створено клас TouchNode.

```
class TouchNode: SCNNode {
    override init() {
        super.init()
        commonInit()
    }
    required init?(coder aDecoder: NSCoder) {
        super.init(coder: aDecoder)
        commonInit()
    }
    func commonInit() {
        let sphere = SCNSphere(radius: 0.01)
        let material = SCNMaterial()
        material.diffuse.contents = UIColor.red
        sphere.firstMaterial = material

        let sphereShape = SCNPhysicsShape(geometry: sphere, options: nil)
        physicsBody = SCNPhysicsBody(type: .kinematic, shape: sphereShape)
    }
}
```

Для використання невидимої сфери, потрібно модифікувати метод startDetection().

```
…
```

```
            let imageFingerPoint = VNImagePointForNormalizedPoint(tipPoint,
Int(self.view.bounds.size.width), Int(self.view.bounds.size.height))
            let hitTestResults = self.sceneView.hitTest(imageFingerPoint, types:
.existingPlaneUsingExtent)
            guard let hitTestResult = hitTestResults.first else { return }
            self.touchNode.simdTransform = hitTestResult.worldTransform
            self.touchNode.position.y += 0.01
            self.touchNode.isHidden = false
        …
        guard let outBuffer = outputBuffer else {
            return
          }
          previewImage = UIImage(ciImage: CIImage(cvPixelBuffer: outBuffer))
          normalizedFingerTip = outBuffer.searchTopPoint()
        …
```

Отже, в результаті тепер користувач має змогу за допомогою своєї руки взаємодіяти з віртуальним об’єктом.

3.4 Висновки

В даному розділі було проведено аналіз мов програмування та IDE. В результаті було прийнято рішення використовувати Xcode та Swift для розробки мобільного додатку. Також були розроблені модулі для роботи нейронною мережею та взаємодії людської руки з віртуальним об’єктом.

4 ТЕСТУВАННЯ ДОДАТКУ

4.1 Вибір методики тестування програми

Перед тим, як розпочати тестування мобільних додатків, головне питання полягає у виборі способів тестування програми. У межах вибору пристрою можна зробити два варіанти: вибір моделі гаджета або вибір між емуляторами. Нижче наведені фактори які потрібно врахувати під час вибору пристрою:

- версія ОС: тестування додатку на всіх стабільних версіях ОС
- роздільна здатність екрана: використання смартфонів з різними екранами
- форм-фактор: якщо додаток сумісний зі смартфонами та планшетами, протестуйте на форм-фактори

Також можна враховувати інші фактори, такі як розмір пам'яті, параметри підключення, тощо.

Не менш важливим критерієм тестування є вибір між фізичним пристроєм та емулятором. На початкових етапах розробки емулятори пристроїв надзвичайно корисні, оскільки допомагають швидкому та ефективному тестуванню. Емулятори пристроїв також економічно вигідні.

Використання емуляторів мобільних пристроїв не означає, що слід взагалі уникати використання фізичних пристроїв. Тестування на фізичних пристроях є обов'язковим – це дозволяє зрозуміти як працюватиме додаток у реальних сценаріях. Тестування мобільних пристроїв – це використання емуляторів та фізичних пристроїв для швидкого та ефективного отримання найкращих результатів. Хмарне тестування мобільних додатків полегшує керування потенційно нескінченними комбінаціями сценаріїв. Хмарні обчислення забезпечують веб-середовище для мобільних тестувань, де можна розгортати, тестувати та керувати програмами.

Одним з важливих факторів є вибір між ручним тестуванням і автоматизованим. Автоматизація є ключовим аспектом для успішного тестування регресії на етапах розробки. Однак автоматизоване тестування

вимагає значної суми початкових інвестицій та додаткового часу на написання сценаріїв. Тому автоматизацію тестування слід проводити лише у сценаріях, коли:

- додаток зростає
- життєвий цикл розвитку мобільних пристроїв довгий
- масштаб і частота регресійного тестування високі
- значна частина тестових випадків включає тестові випадки, які можна отримати

З автоматизованим тестуванням мобільних додатків стає простим:

- перевірка сумісності програми з нещодавно випущеними операційними системами
- перевірка сумісності під час оновлення додатків

В залежності від доступу розробника тестів до коду програми розрізняють тестування «білого ящика» та тестування «чорного ящика»[21].

При тестуванні «білого ящика» (також називають – прозорого ящика), розробник тесту має доступ до коду програми і може написати код, який пов’язаний з бібліотеками програмного забезпечення, яке тестується. Це є типовим для модульного тестування, при якому тестуються тільки окремі частини системи. Воно забезпечує те, що компоненти конструкції є працездатними та стійкими.

При тестуванні «чорного ящика», тестувальник має доступ до програми тільки через ті самі інтерфейси, що й користувач чи замовник, або через зовнішні інтерфейси, які дозволяють іншому комп’ютеру чи процесу підключитись до системи для тестування. Як правило, тестування чорного ящика ведеться з використанням специфікацій або інших документів, які описують вимоги до системи. Зазвичай в даному виді тестування критерій покриття складається з покриття структури вхідних даних, покриття вимог і покриття моделі (в тестуванні на основі моделей).

Для тестування додатку «Hands Gesture AR» було обрано стратегію «чорного ящика» та iPhone X.

4.2 Тестування програмного забезпечення

Протестуємо знаходження людської руки в реальному часі. Для цього необхідно запустити програму, дозволити використовувати камеру смартфону для роботи додатку та помістити ліву руку так, щоб її було видно на екрані смартфону. В результаті в лівому нижньому кутку має з’явитись чорно-біле зображення розпізної руки (рисунок 4.1).

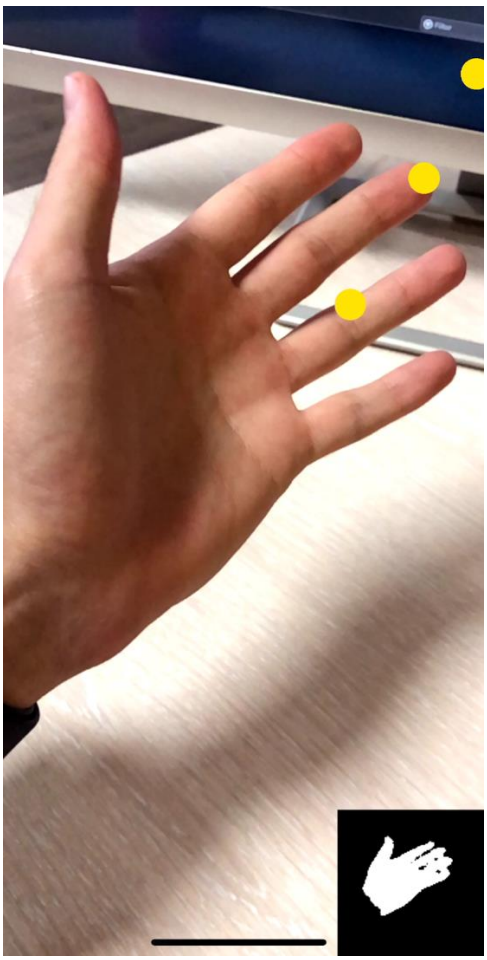


Рисунок 4.1 – Результат знаходження лівої руки

Наступним кроком буде виконання цього ж тесту для правої руки. Система має однаково розпізнавати як ліву так і праву людську руку. Результати

проведеного тестування з знаходженням об’єкта в реальному часі можна побачити на рисунку 4.2.

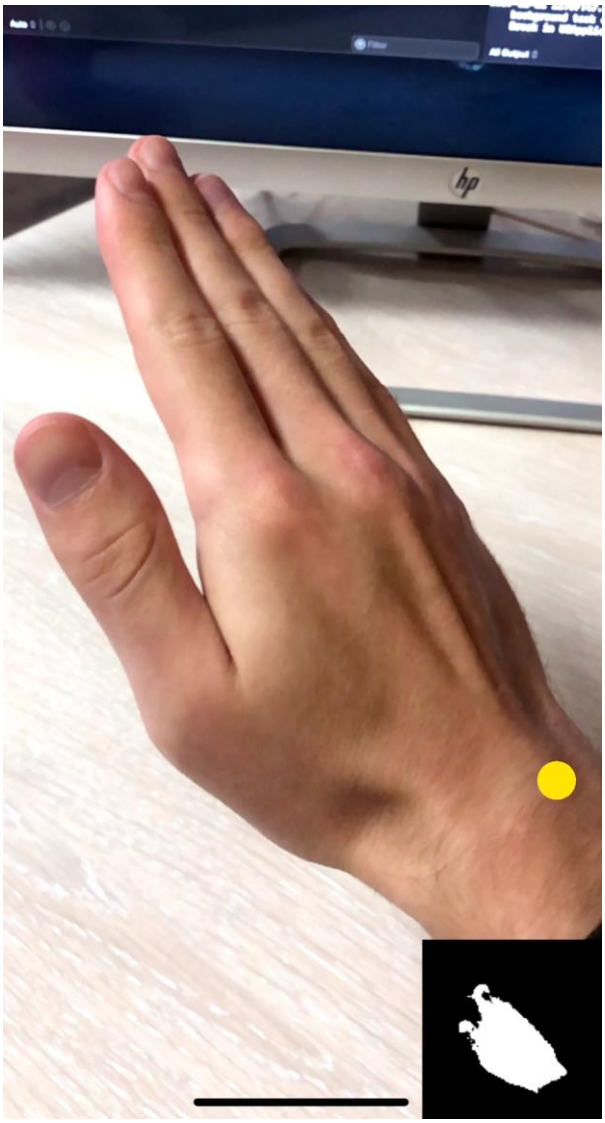


Рисунок 4.2 – Результат знаходження правої руки

Тепер необхідно протестувати створення віртуального об’єкту на поверхні реального світу. Для цього потрібно двічі тапнути на екрани. В результаті на екрані смартфону має з’явитись віртуальний об’єкт у вигляді футбольного м’яча (рисунок 4.3).

Рисунок 4.3 – Результат тесту створення віртуального об’єкта

Під час розробки програмного продукту було створено алгоритм пошуку найвищої точки людської руки. В результаті його виконання на верхній точці руки має з’явитись віртуальна сфера червоного кольору (рисунок 4.4).

Рисунок 4.4 – Результат тестування алгоритму знаходження найвищої точки

Останнім кроком є перевірка можливості взаємодії людської руки з віртуальний об’єктом за допомогою камери смартфону. Результати виконання тесту можна побачити на рисунку 4.5.

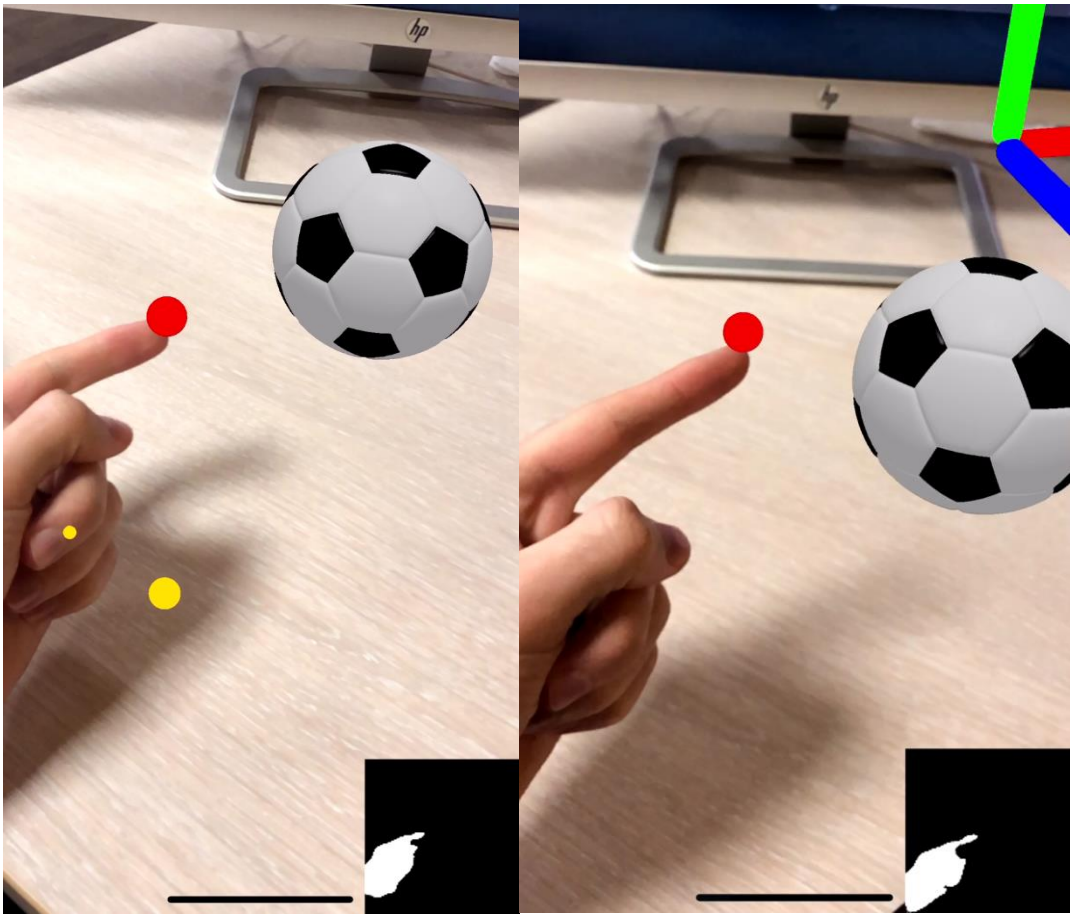


Рисунок 4.5 – Положення м’яча до і після початку взаємодії з рукою

Отже, розроблений програмний додаток працює правильно. Наступним етапом тестування є перевірка споживання ресурсозатратності програмного додатку. Для цього необхідно зайти в Xcode, підключити смартфон до комп’ютера та запустити додаток в режимі debug. Після цього потрібно перейти на вкладку Performance, та перевірити кількість системних ресурсів які використовує програма (рисунок 4.6).

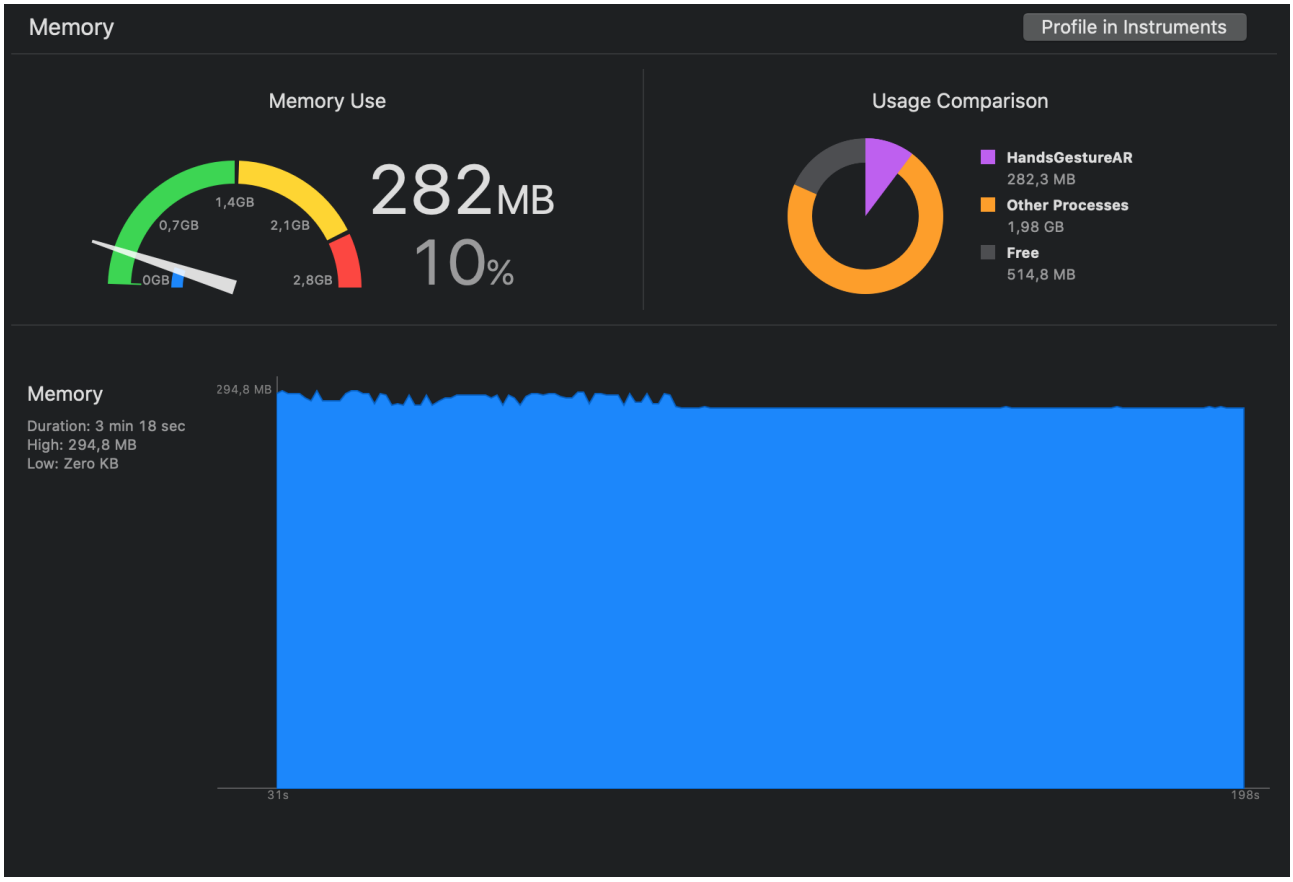


Рисунок 4.6 – Результат тестування ресурсозатратності додатку

Тестування програмного продукту показало повну відповідність поставленому технічному завданню.

4.3 Висновки

Одним з важливих етапів розробки додатку є тестування, так як воно є гарантією якості розроблюваного додатку.

В ході створення даного розділу було розглянуто різні стратегії тестування, та визначено, що для даного додатку найоптимальнішим є тестування «чорного ящика».

При проведенні тестування, отримана повна відповідність вхідних даних і вихідних результатів. Помилок при роботі програми не виявлено і тому вона підтверджує нормальний режим роботи додатку.

5 ЕКОНОМІЧНА ЧАСТИНА

5.1 Оцінювання комерційного потенціалу розробки.

Метою проведення технологічного аудиту є оцінювання комерційного потенціалу розробки методів і програмного забезпечення доповненої реальності для розпізнавання рухів з використанням технологій Swift, ARKit, CoreML.

Для проведення технологічного аудиту залучимо 2-х незалежних експертів. Оцінювання комерційного потенціалу розробки будемо здійснювати за 12-ю критеріями, які містяться в додатку В, та згідно рекомендацій[22].. Результати оцінювання комерційного потенціалу розробки заносимо до таблиці 5.1.

Таблиця 5.1. – Результати оцінювання комерційного потенціалу розробки

Критерії	Експерти	
	к.т.н., доц. Катєльніков Д.І.	к.т.н., доцент Арсенюк І.Р
	Бали, виставлені експертами	
1	2	2
2	3	3
3	4	3
4	3	2
5	2	2
6	3	2
7	2	3
8	2	2
9	3	3
10	3	2
11	3	2
12	2	2
Сума балів	32	28
Середньоарифметична сума балів, СБ	$\overline{СБ} = \frac{\sum_{1}^{3} СБ_i}{2} = 30$	

З отриманих даних таблиці 5.1 видно, що нова розробка має середній рівень комерційного потенціалу. У розділі 1, було проведено порівняння з

аналогами, в результаті якого було виявлено, що найближчим аналогом є додаток ARCore Elements, його головним недоліком є те що користувач може взаємодіяти з віртуальними об’єктами, але тільки через натискання кнопок на смартфоні, в той час як перевагою нової розробки є взаємодія з віртуальними об’єктами за допомогою розпізнавання рухів, що дає змогу взаємодіяти з ними навіть на відстані, якщо людина знадиться в полі зору камери смартфону. Саме тому дана розробка є конкурентоспроможною на ринку. Розроблюваний додаток охоплює широкий спектр користувачів. Програмний продукт буде продаватись в App Store. Для реклами додатку буде використано Google ADS. Комерціалізація на початковому етапі, ведеться пошук партнерів.

5.2 Прогнозування витрат на виконання науково-дослідної роботи та конструкторсько–технологічної роботи

Для розробки нового програмного продукту необхідні такі витрати.

Основна заробітна плата для розробників визначається за формулою (5.1):

$$З_о = \frac{М}{Т_р} \cdot t, \quad (5.1)$$

де М- місячний посадовий оклад конкретного розробника;

$Т_р$ - кількість робочих днів у місяці, $Т_р$ = 22 дні;

t - число днів роботи розробника, t = 25 днів.

Розрахунки заробітних плат для керівника і програміста наведені в таблиці 5.2.

Таблиця 5.2 – Розрахунки основної заробітної плати

Посада	Місячний посадовий оклад, грн.	Оплата за робочий день, грн.	Число днів роботи	Витрати на заробітну плату, грн.
Керівник	20000	950	5	4750
Інженер-програміст	35000	1667	25	41675
Всього:				46425

Розрахуємо додаткову заробітну плату:

$$\text{Здод} = 0{,}1 \cdot 46425 = 4642{,}5(\text{грн.})$$

Нарахування на заробітну плату операторів НЗП розраховується як 37,5…40% від суми їхньої основної та додаткової заробітної плати:

$$\text{Нзп} = (\text{Зо} + \text{Зр}) \cdot \frac{\beta}{100}, \qquad (5.2)$$

$$\text{Нзп} = (46425 + 4642{,}5) \cdot \frac{36{,}3}{100} = 18537{,}50\ (\text{грн.}).$$

Розрахунок амортизаційних витрат для програмного забезпечення виконується за такою формулою:

$$\text{А} = \frac{\text{Ц} \cdot \text{Н}_\text{а}}{100} \cdot \frac{\text{Т}}{12}, \qquad (5.3)$$

де Ц – балансова вартість обладнання, грн;

$\text{Н}_\text{а}$ – річна норма амортизаційних відрахувань % (для програмного забезпечення 25%);

Т – Термін використання (Т=3 міс.).

Таблиця 5.3 – Розрахунок амортизаційних відрахувань

Найменування	Ціна, грн.	Норма амортизації, %	Термін використання, м.	Сума амортизації
Персональний комп’ютер	20000	20	1	333
Смартфон	15000	20	1	250
Всього	583			

Розрахуємо витрати на комплектуючі. Витрати на комплектуючі розрахуємо за формулою:

$$K = \sum_1^n Н_і \cdot Ц_і \cdot К_і, \quad (5.4)$$

де n – кількість комплектуючих;

Ні - кількість комплектуючих і-го виду;

Ці – покупна ціна комплектуючих і-го виду, грн;

Кі – коефіцієнт транспортних витрат (приймемо Кі = 1,1).

Таблиця 5.4 - Витрати на комплектуючі, що були використані для розробки ПЗ.

Найменування матеріалу	Одиниці виміру	Ціна, грн.	Витрачено	Вартість витрачених матеріалів, грн.
Флешка	шт.	180	1	180
Пачка паперу	уп.	115	1	115
Ручка	шт.	5	1	5
Всього з урахуванням транспортних витрат				330

Витрати на силову електроенергію розраховуються за формулою:

$$В_е = В \cdot П \cdot Ф \cdot К_п ; \quad (5.5)$$

де В – вартість 1кВт-години електроенергії (В=1,7 грн/кВт);

П – установлена потужність комп’ютера (П=0,6кВт);

Ф – фактична кількість годин роботи комп’ютера (Ф=100 год.);

$К_п$ – коефіцієнт використання потужності ($К_п < 1$, $К_п = 0,7$).

$$Ве = 1,7 \cdot 0,6 \cdot 100 \cdot 0,7 = 71,4 \text{ (грн.)}$$

Розрахуємо інші витрати $В_{ін}$.

Інші витрати $І_в$ можна прийняти як (100…300)% від суми основної заробітної плати розробників та робітників, які були виконували дану роботу, тобто:

$$В_{ін} = (1..3) \cdot (З_о + З_р). \quad (5.6)$$

$$В_{ін} = 1 \cdot (46425 + 4642,5) = 51067,5 \text{ грн.}.$$

Сума всіх попередніх статей витрат дає витрати на виконання даної частини роботи:

$$В = З_о + З_д + Н_{зп} + А + К + В_е + І_в$$

В = 46425+4642,5+18537,50+583+330+71,4+51067,5 = 121656,9 (грн.)

Розрахуємо загальну вартість наукової роботи $B_{заг}$ за формулою:

$$В_{заг} = \frac{В_{ін}}{\alpha} \quad (5.7)$$

де α – частка витрат, які безпосередньо здійснює виконавець даного етапу роботи, у відн. одиницях = 1.

$$В_{заг} = \frac{121656,9}{1} = 121656,9$$

Прогнозування загальних витрат ЗВ на виконання та впровадження результатів виконаної наукової роботи здійснюється за формулою:

$$ЗВ = \frac{В_{заг}}{\beta} \quad (5.8)$$

де β – коефіцієнт, який характеризує етап (стадію) виконання даної роботи.

Отже, розрахуємо загальні витрати:

$$ЗВ = \frac{121656,9}{0,9} = 135174,33 \text{ (грн.)}$$

5.3 Прогнозування комерційних ефектів від реалізації результатів розробки

Спрогнозуємо отримання прибутку від реалізації результатів нашої розробки. Зростання чистого прибутку можна оцінити у теперішній вартості грошей. Це забезпечить підприємству (організації) надходження додаткових коштів, які дозволять покращити фінансові результати діяльності .

Оцінка зростання чистого прибутку підприємства від впровадження результатів наукової розробки. У цьому випадку збільшення чистого прибутку підприємства $\Delta П_і$ для кожного із років, протягом яких очікується отримання позитивних результатів від впровадження розробки, розраховується за формулою:

$$\Delta П_i = \sum_1^n (\Delta П_я \cdot N + П_я \Delta N)_i \quad (5.9)$$

де $\Delta П_я$ – покращення основного якісного показника від впровадження результатів розробки у даному році;

N – основний кількісний показник, який визначає діяльність підприємства у даному році до впровадження результатів наукової розробки;

ΔN – покращення основного кількісного показника діяльності підприємства від впровадження результатів розробки;

$П_я$ – основний якісний показник, який визначає діяльність підприємства у даному році після впровадження результатів наукової розробки;

n – кількість років, протягом яких очікується отримання позитивних результатів від впровадження розробки.

В результаті впровадження результатів наукової розробки витрати на виготовлення програмного продукту зменшиться на 30 грн (що автоматично спричинить збільшення чистого прибутку підприємства на 30 грн), а кількість користувачів, які будуть користуватись збільшиться: протягом першого року – на 4000 користувачів, протягом другого року – на 350 користувачів, протягом третього року – 300 користувачів. Реалізація програмного продукту до

впровадження результатів наукової розробки складала 1 користувачів, а прибуток, що отримував розробник до впровадження результатів наукової розробки – 30 грн.

Спрогнозуємо збільшення чистого прибутку від впровадження результатів наукової розробки у кожному році відносно базового.

Отже, збільшення чистого продукту $\Delta П_1$ протягом першого року складатиме:

$$\Delta П_1 = 30 \cdot 1 + (30 + 30) \cdot 4000 = 240060 \text{ грн.}$$

Протягом другого року:

$$\Delta П_2 = 30 \cdot 1 + (30 + 30) \cdot (4000 + 350) = 261030 \text{ грн.}$$

Протягом третього року:

$$\Delta П_3 = 30 \cdot 1 + (30 + 30) \cdot (4000 + 350 + 300) = 279030 \text{ грн.}$$

5.4. Розрахунок ефективності вкладених інвестицій та період їх окупності

Визначимо абсолютну і відносну ефективність вкладених інвестором інвестицій та розрахуємо термін окупності.

Абсолютна ефективність $Е_{абс}$вкладених інвестицій розраховується за формулою:

$$Е_{абс} = (ПП - PV), \qquad (5.10)$$

де $\Delta П_i$ – збільшення чистого прибутку у кожному із років, протягом яких виявляються результати виконаної та впровадженої НДДКР, грн;

т – період часу, протягом якого виявляються результати впровадженої НДДКР, 3 роки;

τ – ставка дисконтування, за яку можна взяти щорічний прогнозований рівень інфляції в країні; для України цей показник знаходиться на рівні 0,1;

t – період часу (в роках) від моменту отримання чистого прибутку до точки 2, 3,4.

Рисунок, що характеризує рух платежів (інвестицій та додаткових прибутків) буде мати вигляд, рисунок 5.1.

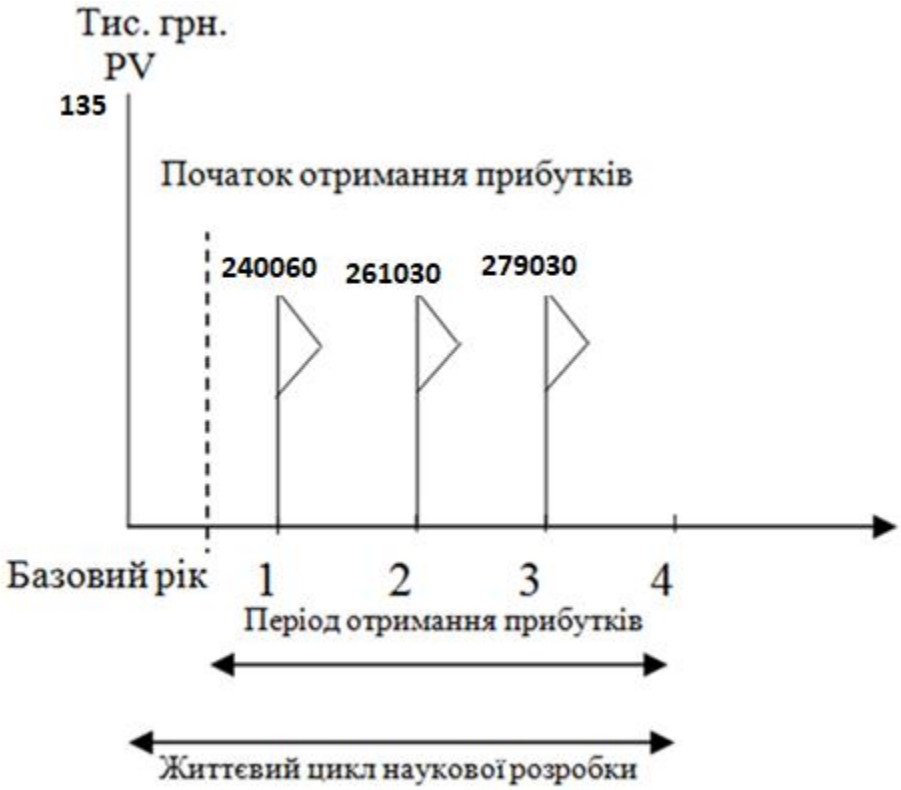


Рисунок 5.1 – Вісь часу з фіксацією платежів, що мають місце під час розробки та впровадження результатів НДДКР

Розрахуємо вартість чистих прибутків за формулою:

$$ПП = \sum_1^m \frac{\Delta П_i}{(1+\tau)^t} \quad (5.11)$$

де $\Delta П_i$ – збільшення чистого прибутку у кожному із років, протягом яких виявляються результати виконаної та впровадженої НДДКР, грн;

т – період часу, протягом якого виявляються результати впровадженої НДДКР, роки;

τ – ставка дисконтування, за яку можна взяти щорічний прогнозований рівень інфляції в країні; для України цей показник знаходиться на рівні 0,1;

t – період часу (в роках) від моменту отримання чистого прибутку до точки.

Отже, розрахуємо вартість чистого прибутку:

$$ПП = \frac{135174,33}{(1+0,1)^0} + \frac{240060}{(1+0,1)^2} + \frac{261030}{(1+0,1)^3} = 529834,18(грн.)$$

Тоді розрахуємо $E_{абс}$:

$$Е_{абс} = 529834,18 - 135174,33 = 394659,85 \text{ грн.}$$

Оскільки $E_{абс} > 0$, то вкладання коштів на виконання та впровадження результатів НДДКР буде доцільним.

Розрахуємо відносну (щорічну) ефективність вкладених в наукову розробку інвестицій $Е_в$ за формулою:

$$Е_в = \sqrt[Т]{1 + \frac{Е_{абс}}{PV}} - 1 \qquad (5.12)$$

де $Е_{абс}$ – абсолютна ефективність вкладених інвестицій, грн;

PV – теперішня вартість інвестицій PV = ЗВ, грн;

$Т_ж$ – життєвий цикл наукової розробки, роки.

Тоді будемо мати:

$$Е_в = \sqrt[3]{1 + \frac{394659,85}{135174,33}} - 1 = 1,57 \text{ або } 157\ \%$$

Далі, розраховану величина $Е_в$ порівнюємо з мінімальною (бар'єрною) ставкою дисконтування $\tau_{мін}$, яка визначає ту мінімальну дохідність, нижче за яку інвестиції вкладатися не будуть. У загальному вигляді мінімальна (бар'єрна) ставка дисконтування $\tau_{мін}$ визначається за формулою:

$$\tau = d + f,$$

де d – середньозважена ставка за депозитними операціями в комерційних банках; в 2019 році в Україні d = 0,2;

f – показник, що характеризує ризикованість вкладень, величина f = 0,1.

$$\tau = 0{,}2 + 0{,}1 = 0{,}3$$

Оскільки $E_в$ = 157% > $\tau_{мін}$ = 0,3 = 30%, то у інвестор буде зацікавлений вкладати гроші в дану наукову розробку.

Термін окупності вкладених у реалізацію наукового проекту інвестицій. Термін окупності вкладених у реалізацію наукового проекту інвестицій $T_{ок}$ розраховується за формулою:

$$T_{ок} = \frac{1}{E_в}$$

$$T_{ок} = \frac{1}{1{,}57} = 0{,}63 \text{ року}$$

Обрахувавши термін окупності даної наукової розробки, можна зробити висновок, що фінансування даної наукової розробки буде доцільним.

5.5 Висновки

Для проведення технологічного аудиту було залучено 2-х експертів, в результаті чого було виявлено, що розробка має середній рівень комерційного потенціалу. Також розраховано загальні витрати на розробку додатку, які становлять 135174,33 грн., Абсолютна ефективність вкладених інвестицій дорівнює 394659,85 грн., а відносна ефективність 157%, саме тому вкладання коштів на виконання та впровадження результатів НДДКР буде доцільним. Також в результаті виконання обрахунків, було виявлено, що термін окупності проекту становить 0.63 року.

ВИСНОВКИ

В ході виконання магістерської кваліфікаційної роботи було виконано аналіз основних аналогів, серед функцій яких є розпізнавання рухів людської руки, виявлено їхні переваги та недоліки. На основі отриманих результатів було прийнято рішення про створення власного методу розпізнавання рухів, який би вирішував проблеми що присутні в аналогах.

Було запропоновано метод розпізнавання рухів руки, особливість якого полягає у поєднанні доповненої реальності та нейронної мережі, а також створення алгоритму для знаходження точки руки. Таким чином було розроблено метод який не лише розпізнає рухи, а й дозволяє взаємодіяти за допомогою руки з віртуальними об’єктами лише за допомогою камери смартфону.

Розроблено мобільний додаток який дозволяє розпізнавати рухи людської руки та взаємодіяти з віртуальним футбольним м’ячем. Технологіями для реалізації системи було обрано: Swift, ARKit, CoreML.

Тестування програми довело повну працездатність даного програмного продукту та відповідність поставленому технічному завданню.

В ході економічного обґрунтування розробки виконано аналіз комерційного потенціалу, розраховано витрати на виконання науково-дослідної, дослідно-конструкторської та конструкторсько-технологічної роботи. Також було виконано розрахунок мінімальної ціни та чистого прибутку, в результаті чого розробку було визнано економічно доцільною.

ПЕРЕЛІК ПОСИЛАНЬ

1. Смартфони [Електронний ресурс]. – Режим доступу: https://uk.wikipedia.org/wiki/%D0%A1%D0%BC%D0%B0%D1%80%D1%82%D1%84%D0%BE%D0%BD

2. BLE [Електронний ресурс]. – Режим доступу: https://developer.apple.com/library/archive/documentation/NetworkingInternetWeb/Conceptual/CoreBluetooth_concepts/CoreBluetoothBackgroundProcessingForIOSApps/PerformingTasksWhileYourAppIsInTheBackground.html#//apple_ref/doc/uid/TP40013257-CH7-SW7

3. Галушкин А.И. Теория нейронных сетей. / А.И. Галушкин - М.: ИПРЖР, 2010. – 348 с.

4. TensorFlow [Електронний ресурс]. – Режим доступу: https://ru.wikipedia.org/wiki/TensorFlow

5. CoreML [Електронний ресурс]. – Режим доступу: https://developer.apple.com/documentation/coreml

6. ARKit [Електронний ресурс]. – Режим доступу: https://developer.apple.com/augmented-reality/

7. ARCore Elements [Електронний ресурс]. – Режим доступу: https://play.google.com/store/apps/details?id=com.google.ar.unity.ddelements&hl=ru

8. VR Gesture Player [Електронний ресурс]. – Режим доступу: https://play.google.com/store/apps/details?id=com.macron.gestureplayer&hl=ru

9. Wanna Nails [Електронний ресурс]. – Режим доступу: https://apps.apple.com/ru/app/wanna-nails/id1334108416

10. Face Racer – no hands! [Електронний ресурс]. – Режим доступу: https://apps.apple.com/us/app/face-racer-no-hands/id1448620017

11. Gao L. et al. Real-time visual representations for mobile mixed reality remote collaboration //SIGGRAPH Asia 2018 Virtual & Augmented Reality. – ACM, 2018. – С. 15.

12. Алгоритм и методы обнаружения жестов руки [Електронний ресурс]. – Режим доступу: https://cyberleninka.ru/article/n/algoritm-i-metody-obnaruzheniya-i-raspoznavaniya-zhestov-ruki-na-video-v-rezhime-realnogo-vremeni

13. Turi Create [Електронний ресурс]. – Режим доступу: https://medium.com/slalom-engineering/object-detection-training-with-apples-turi-create-for-coreml-2019-update-fa1c015f2366

14. PyTorch [Електронний ресурс]. – Режим доступу: https://habr.com/ru/post/334380/

15. Jupyter Notebook [Електронний ресурс]. – Режим доступу: https://jupyter.org/about

16. Objective-C [Електронний ресурс]. – Режим доступу: https://uk.wikipedia.org/wiki/Objective-C

17. Swift [Електронний ресурс]. – Режим доступу: https://habr.com/ru/hub/swift/

18. Мова C# [Електронний ресурс]. – Режим доступу: https://uk.wikipedia.org/wiki/C_Sharp

19. Мова Python [Електронний ресурс]. – Режим доступу: https://ru.wikipedia.org/wiki/Python

20. Мова C++ [Електронний ресурс]. – Режим доступу: https://ru.wikipedia.org/wiki/C%2B%2B

21. Степанченко И.В. Методы тестирования программного обеспечения : учебное пособие / И.В. Степанченко. – Волгоград : ВолгГТУ, 2006. – 74с.

22. Методичні вказівки до виконання студентами-магістрантами економічної частини магістерських кваліфікаційних робіт / Уклад. В. О. Козловський – Вінниця: ВНТУ, 2012. – 22 с.

ДОДАТКИ

Додаток А. Технічне завдання

Міністерство освіти і науки України
Вінницький національний технічний університет
Факультет інформаційних технологій та комп'ютерної інженерії

ЗАТВЕРДЖУЮ
д.т.н., проф. О. Н. Романюк
"____ "_____2019 р.

Технічне завдання
на магістерську кваліфікаційну роботу
«Розробка методів і програмного забезпечення доповненої реальності для розпізнавання рухів з використанням технологій Swift, ARKit, CoreML»
за спеціальністю
121 – Інженерія програмного забезпечення

Керівник магістерської кваліфікаційної роботи:
_____к.т.н., доцент кафедри ПЗ, Катєльніков Д.І.
«_____»_____2019 р.
Виконав:
_____студент гр. 2ПІ-18м, Туйчев В.В.
«_____»_____2019 р.

Вінниця ВНТУ 2019

1. Найменування та галузь застосування

Магістерська кваліфікаційна робота: «Розробка методів і програмного забезпечення доповненої реальності для розпізнавання рухів з використанням технологій Swift, ARKit, CoreML».

Галузь застосування - системи комп'ютерної графіки.

2. Підстава для розробки.

Підставою для виконання магістерської кваліфікаційної роботи (МКР) є індивідуальне завдання на МКР та наказ № ректора по ВНТУ про закріплення тем МКР.

3. Мета та призначення розробки.

Метою роботи є покращення взаємодії користувач-комп'ютер за рахунок поєднання методів створення доповненої реальності і розпізнавання рухів.

Призначення роботи – розробка методів і програмного забезпечення доповненої реальності для розпізнавання рухів та інтерактивної взаємодії з віртуальними об'єктами.

4. Вихідні дані для проведення НДР

Перелік основних літературних джерел, на основі яких буде виконуватись МКР.

1. Buerli M., Misslinger S. Introducing ARKit-Augmented Reality for iOS //Apple Worldwide Developers Conference (WWDC'17). – 2017. – С. 1-187.

2. Nugroho A., Pramono B. A. Aplikasi Mobile Augmented Reality Berbasis Vuforia Dan Unity Pada Pengenalan Objek 3D Dengan Studi Kasus Gedung M Universitas Semarang //Jurnal Transformatika. – 2017. – Т. 14. – №. 2. – С. 86-91.

3. Allan A. Learning iPhone Programming: From Xcode to App Store. – " O'Reilly Media, Inc.", 2010.

4. Тархов Д.А. Нейронные сети. Модели и алгоритмы. – М.: Радиотехника, 2010. – 82 с.

5. Технічні вимоги

Мова програмування: Swift;

Середовище розробки: Xcode;

Бібліотеки: Swift, ARKit, CoreML, PyTorch.

6. Конструктивні вимоги.

Мобільний додаток повинен відповідати всім вимогам, повинен бути зручним та зрозумілим у використанні.

Графічна та текстова документація повинна відповідати діючим стандартам України.

7. Перелік технічної документації, що пред'являється по закінченню робіт:

– пояснювальна записка до МКР;

– технічне завдання;

– лістинги програми.

8. Вимоги до рівня уніфікації та стандартизації

При розробці програмних засобів слід дотримуватися уніфікації і ДСТУ.

9. Стадії та етапи розробки:

№ з/п	Назва етапів магістерської кваліфікаційної роботи	Строк виконання етапів роботи
1	Аналіз стану проблеми та порівняльний аналіз аналогів	01.10.19 – 11.10.19
2	Розробка методів інтерактивної взаємодії користувач-комп'ютер	12.10.19 – 17.10.19
3	Розробка програмного забезпечення	18.10.19 – 27.11.19
4	Тестування програмної системи	28.11.19 – 30.11.19
5	Економічна частина	01.12.19 – 08.12.19

10. Порядок контролю та прийняття.

Виконання етапів магістерської кваліфікаційної роботи контролюється керівником згідно з графіком виконання роботи.

Прийняття магістерської кваліфікаційної роботи здійснюється ДЕК, затвердженою зав. кафедрою згідно з графіком

Додаток Б. Програмний код додатку

```
import Foundation
import SceneKit
import UIKit
import ARKit
class MainViewController: UIViewController {
	var dragOnInfinitePlanesEnabled = false
	var currentGesture: Gesture?
	var use3DOFTrackingFallback = false
	var screenCenter: CGPoint?

	let session = ARSession()
	var sessionConfig: ARConfiguration = ARWorldTrackingConfiguration()

	var trackingFallbackTimer: Timer?

	// Use average of recent virtual object distances to avoid rapid changes in
object scale.
	var recentVirtualObjectDistances = [CGFloat]()

	let DEFAULT_DISTANCE_CAMERA_TO_OBJECTS = Float(10)

	override func viewDidLoad() {
	 super.viewDidLoad()

	 Setting.registerDefaults()
	 setupScene()
	 setupDebug()
	 setupUIControls()
		setupFocusSquare()
		updateSettings()
		resetVirtualObject()
    }

	override func viewDidAppear(_ animated: Bool) {
		super.viewDidAppear(animated)

		UIApplication.shared.isIdleTimerDisabled = true
		restartPlaneDetection()
	}

	override func viewWillDisappear(_ animated: Bool) {
		super.viewWillDisappear(animated)
		session.pause()
```

```
        }

    // MARK: - ARKit / ARSCNView
    var use3DOFTracking = false {
            didSet {
                    if use3DOFTracking {
                            sessionConfig = ARWorldTrackingConfiguration()
                    }
                    sessionConfig.isLightEstimationEnabled =
UserDefaults.standard.bool(for: .ambientLightEstimation)
                    session.run(sessionConfig)
            }
        }
        @IBOutlet var sceneView: ARSCNView!

    // MARK: - Ambient Light Estimation
        func toggleAmbientLightEstimation(_ enabled: Bool) {
         if enabled {
                    if !sessionConfig.isLightEstimationEnabled {
                            sessionConfig.isLightEstimationEnabled = true
                            session.run(sessionConfig)
                    }
         } else {
                    if sessionConfig.isLightEstimationEnabled {
                            sessionConfig.isLightEstimationEnabled = false
                            session.run(sessionConfig)
                    }
         }
    }

    // MARK: - Virtual Object Loading
        var isLoadingObject: Bool = false {
                didSet {
                        DispatchQueue.main.async {
                                self.settingsButton.isEnabled = !self.isLoadingObject
                                self.addObjectButton.isEnabled = !self.isLoadingObject
                                self.screenshotButton.isEnabled = !self.isLoadingObject
                                self.restartExperienceButton.isEnabled =
!self.isLoadingObject
                        }
                }
        }

        @IBOutlet weak var addObjectButton: UIButton!

        @IBAction func chooseObject(_ button: UIButton) {
```

```
            // Abort if we are about to load another object to avoid concurrent
modifications of the scene.
            if isLoadingObject { return }

            textManager.cancelScheduledMessage(forType: .contentPlacement)

            let rowHeight = 45
            let popoverSize = CGSize(width: 250, height: rowHeight *
VirtualObjectSelectionViewController.COUNT_OBJECTS)

            let objectViewController = VirtualObjectSelectionViewController(size:
popoverSize)
            objectViewController.delegate = self
            objectViewController.modalPresentationStyle = .popover
            objectViewController.popoverPresentationController?.delegate = self
            self.present(objectViewController, animated: true, completion: nil)

            objectViewController.popoverPresentationController?.sourceView = button
            objectViewController.popoverPresentationController?.sourceRect =
button.bounds
    }

    // MARK: - Planes
       var planes = [ARPlaneAnchor: Plane]()

    func addPlane(node: SCNNode, anchor: ARPlaneAnchor) {

            let pos = SCNVector3.positionFromTransform(anchor.transform)
            textManager.showDebugMessage("NEW SURFACE DETECTED AT
\(pos.friendlyString())")

            let plane = Plane(anchor, showDebugVisuals)

            planes[anchor] = plane
            node.addChildNode(plane)

            textManager.cancelScheduledMessage(forType: .planeEstimation)
            textManager.showMessage("SURFACE DETECTED")
            if !VirtualObjectsManager.shared.isAVirtualObjectPlaced() {
                   textManager.scheduleMessage("TAP + TO PLACE AN OBJECT",
inSeconds: 7.5, messageType: .contentPlacement)
            }
       }

       func restartPlaneDetection() {
            // configure session
```

```
            if let worldSessionConfig = sessionConfig as?
ARWorldTrackingConfiguration {
                    worldSessionConfig.planeDetection = .horizontal
                    session.run(worldSessionConfig, options: [.resetTracking,
.removeExistingAnchors])
            }

            // reset timer
            if trackingFallbackTimer != nil {
                    trackingFallbackTimer!.invalidate()
                    trackingFallbackTimer = nil
            }

            textManager.scheduleMessage("FIND A SURFACE TO PLACE AN OBJECT",
inSeconds: 7.5, messageType: .planeEstimation)
      }

    // MARK: - Focus Square
    var focusSquare: FocusSquare?

    func setupFocusSquare() {
            focusSquare?.isHidden = true
            focusSquare?.removeFromParentNode()
            focusSquare = FocusSquare()
            sceneView.scene.rootNode.addChildNode(focusSquare!)

            textManager.scheduleMessage("TRY MOVING LEFT OR RIGHT", inSeconds: 5.0,
messageType: .focusSquare)
    }

       func updateFocusSquare() {
            guard let screenCenter = screenCenter else { return }

            let virtualObject =
VirtualObjectsManager.shared.getVirtualObjectSelected()
            if virtualObject != nil && sceneView.isNode(virtualObject!,
insideFrustumOf: sceneView.pointOfView!) {
                    focusSquare?.hide()
            } else {
                    focusSquare?.unhide()
            }
            let (worldPos, planeAnchor, _) =
worldPositionFromScreenPosition(screenCenter, objectPos: focusSquare?.position)
            if let worldPos = worldPos {
                    focusSquare?.update(for: worldPos, planeAnchor: planeAnchor,
camera: self.session.currentFrame?.camera)
```

```
                        textManager.cancelScheduledMessage(forType: .focusSquare)
                }
        }

        // MARK: - Hit Test Visualization
        var hitTestVisualization: HitTestVisualization?

        var showHitTestAPIVisualization = UserDefaults.standard.bool(for:
.showHitTestAPI) {
                didSet {
                        UserDefaults.standard.set(showHitTestAPIVisualization, for:
.showHitTestAPI)
                        if showHitTestAPIVisualization {
                                hitTestVisualization = HitTestVisualization(sceneView:
sceneView)
                        } else {
                                hitTestVisualization = nil
                        }
                }
        }

    // MARK: - Debug Visualizations
        @IBOutlet var featurePointCountLabel: UILabel!

        func refreshFeaturePoints() {
                guard showDebugVisuals else {
                        return
                }

                guard let cloud = session.currentFrame?.rawFeaturePoints else {
                        return
                }

                DispatchQueue.main.async {
                        self.featurePointCountLabel.text = "Features:
\(cloud.__count)".uppercased()
                }
        }

    var showDebugVisuals: Bool = UserDefaults.standard.bool(for: .debugMode) {
         didSet {
                        featurePointCountLabel.isHidden = !showDebugVisuals
                        debugMessageLabel.isHidden = !showDebugVisuals
                        messagePanel.isHidden = !showDebugVisuals
                        planes.values.forEach {
$0.showDebugVisualization(showDebugVisuals) }
```

```
                    sceneView.debugOptions = []
                    if showDebugVisuals {
                            sceneView.debugOptions =
[ARSCNDebugOptions.showFeaturePoints, ARSCNDebugOptions.showWorldOrigin]
                    }
            UserDefaults.standard.set(showDebugVisuals, for: .debugMode)
        }
    }

    func setupDebug() {
            messagePanel.layer.cornerRadius = 3.0
            messagePanel.clipsToBounds = true
    }

    // MARK: - UI Elements and Actions
       @IBOutlet weak var messagePanel: UIView!
       @IBOutlet weak var messageLabel: UILabel!
       @IBOutlet weak var debugMessageLabel: UILabel!

       var textManager: TextManager!

    func setupUIControls() {
            textManager = TextManager(viewController: self)
            debugMessageLabel.isHidden = true
            featurePointCountLabel.text = ""
            debugMessageLabel.text = ""
            messageLabel.text = ""
    }

       @IBOutlet weak var restartExperienceButton: UIButton!
       var restartExperienceButtonIsEnabled = true

       @IBAction func restartExperience(_ sender: Any) {
            guard restartExperienceButtonIsEnabled, !isLoadingObject else {
                    return
            }

            DispatchQueue.main.async {
                    self.restartExperienceButtonIsEnabled = false

                    self.textManager.cancelAllScheduledMessages()
                    self.textManager.dismissPresentedAlert()
                    self.textManager.showMessage("STARTING A NEW SESSION")
                    self.use3DOFTracking = false

                    self.setupFocusSquare()
```

```
//                      self.loadVirtualObject()
                        self.restartPlaneDetection()


        self.restartExperienceButton.setImage(#imageLiteral(resourceName: "restart"),
for: [])

                        // Disable Restart button for five seconds in order to give the
session enough time to restart.
                        DispatchQueue.main.asyncAfter(deadline: .now() + 5.0, execute: {
                                self.restartExperienceButtonIsEnabled = true
                        })
                }
        }

        @IBOutlet weak var screenshotButton: UIButton!
        @IBAction func takeSnapShot() {
                guard sceneView.session.currentFrame != nil else { return }
                focusSquare?.isHidden = true

                let imagePlane = SCNPlane(width: sceneView.bounds.width / 6000, height:
sceneView.bounds.height / 6000)
                imagePlane.firstMaterial?.diffuse.contents = sceneView.snapshot()
                imagePlane.firstMaterial?.lightingModel = .constant

                let planeNode = SCNNode(geometry: imagePlane)
                sceneView.scene.rootNode.addChildNode(planeNode)

                focusSquare?.isHidden = false
        }

        // MARK: - Settings
        @IBOutlet weak var settingsButton: UIButton!

        @IBAction func showSettings(_ button: UIButton) {
                let storyboard = UIStoryboard(name: "Main", bundle: nil)
                guard let settingsViewController =
storyboard.instantiateViewController(
                        withIdentifier: "settingsViewController") as?
SettingsViewController else {
                        return
                }

                let barButtonItem = UIBarButtonItem(barButtonSystemItem: .done, target:
self, action: #selector(dismissSettings))
```

```
            settingsViewController.navigationItem.rightBarButtonItem =
barButtonItem
            settingsViewController.title = "Options"

            let navigationController = UINavigationController(rootViewController:
settingsViewController)
            navigationController.modalPresentationStyle = .popover
            navigationController.popoverPresentationController?.delegate = self
            navigationController.preferredContentSize = CGSize(width:
sceneView.bounds.size.width - 20,
                                                               height:
sceneView.bounds.size.height - 50)
            self.present(navigationController, animated: true, completion: nil)

            navigationController.popoverPresentationController?.sourceView =
settingsButton
            navigationController.popoverPresentationController?.sourceRect =
settingsButton.bounds
        }

    @objc
    func dismissSettings() {
            self.dismiss(animated: true, completion: nil)
            updateSettings()
        }

        private func updateSettings() {
            let defaults = UserDefaults.standard

            showDebugVisuals = defaults.bool(for: .debugMode)
            toggleAmbientLightEstimation(defaults.bool(for:
.ambientLightEstimation))
            dragOnInfinitePlanesEnabled = defaults.bool(for: .dragOnInfinitePlanes)
            showHitTestAPIVisualization = defaults.bool(for: .showHitTestAPI)
            use3DOFTracking       = defaults.bool(for: .use3DOFTracking)
            use3DOFTrackingFallback = defaults.bool(for: .use3DOFFallback)
            for (_, plane) in planes {
                plane.updateOcclusionSetting()
            }
        }

        // MARK: - Error handling
        func displayErrorMessage(title: String, message: String, allowRestart: Bool =
false) {
            textManager.blurBackground()
```

```
            if allowRestart {
                let restartAction = UIAlertAction(title: "Reset", style:
.default) { _ in
                    self.textManager.unblurBackground()
                    self.restartExperience(self)
                }
                textManager.showAlert(title: title, message: message, actions:
[restartAction])
            } else {
                textManager.showAlert(title: title, message: message, actions:
[])
            }
        }
}

// MARK: - ARKit / ARSCNView
extension MainViewController {
        func setupScene() {
            sceneView.setUp(viewController: self, session: session)
            DispatchQueue.main.async {
                self.screenCenter = self.sceneView.bounds.mid
            }
        }

        func session(_ session: ARSession, cameraDidChangeTrackingState camera:
ARCamera) {
            textManager.showTrackingQualityInfo(for: camera.trackingState,
autoHide: !self.showDebugVisuals)

            switch camera.trackingState {
            case .notAvailable:
                textManager.escalateFeedback(for: camera.trackingState,
inSeconds: 5.0)
            case .limited:
                if use3DOFTrackingFallback {
                    // After 10 seconds of limited quality, fall back to
3DOF mode.
                    trackingFallbackTimer =
Timer.scheduledTimer(withTimeInterval: 10.0, repeats: false, block: { _ in
                        self.use3DOFTracking = true
                        self.trackingFallbackTimer?.invalidate()
                        self.trackingFallbackTimer = nil
                    })
                } else {
                    textManager.escalateFeedback(for: camera.trackingState,
inSeconds: 10.0)
```

```
                        }
                case .normal:
                        textManager.cancelScheduledMessage(forType:
.trackingStateEscalation)
                        if use3DOFTrackingFallback && trackingFallbackTimer != nil {
                                trackingFallbackTimer!.invalidate()
                                trackingFallbackTimer = nil
                        }
                }
        }

        func session(_ session: ARSession, didFailWithError error: Error) {
                guard let arError = error as? ARError else { return }

                let nsError = error as NSError
                var sessionErrorMsg = "\(nsError.localizedDescription)
\(nsError.localizedFailureReason ?? "")"
                if let recoveryOptions = nsError.localizedRecoveryOptions {
                        for option in recoveryOptions {
                                sessionErrorMsg.append("\(option).")
                        }
                }

                let isRecoverable = (arError.code == .worldTrackingFailed)
                if isRecoverable {
                        sessionErrorMsg += "\nYou can try resetting the session or quit
the application."
                } else {
                        sessionErrorMsg += "\nThis is an unrecoverable error that
requires to quit the application."
                }

                displayErrorMessage(title: "We're sorry!", message: sessionErrorMsg,
allowRestart: isRecoverable)
        }

        func sessionWasInterrupted(_ session: ARSession) {
                textManager.blurBackground()
                textManager.showAlert(title: "Session Interrupted",
                                      message: "The session will be reset after the
interruption has ended.")
        }

        func sessionInterruptionEnded(_ session: ARSession) {
                textManager.unblurBackground()
```

```
                session.run(sessionConfig, options: [.resetTracking,
.removeExistingAnchors])
                restartExperience(self)
                textManager.showMessage("RESETTING SESSION")
        }
}

// MARK: Gesture Recognized
extension MainViewController {
        override func touchesBegan(_ touches: Set<UITouch>, with event: UIEvent?) {
                guard let object =
VirtualObjectsManager.shared.getVirtualObjectSelected() else {
                        return
                }

                if currentGesture == nil {
                        currentGesture = Gesture.startGestureFromTouches(touches,
self.sceneView, object)
                } else {
                        currentGesture =
currentGesture!.updateGestureFromTouches(touches, .touchBegan)
                }

                displayVirtualObjectTransform()
        }

        override func touchesMoved(_ touches: Set<UITouch>, with event: UIEvent?) {
                if !VirtualObjectsManager.shared.isAVirtualObjectPlaced() {
                        return
                }
                currentGesture = currentGesture?.updateGestureFromTouches(touches,
.touchMoved)
                displayVirtualObjectTransform()
        }

        override func touchesEnded(_ touches: Set<UITouch>, with event: UIEvent?) {
                if !VirtualObjectsManager.shared.isAVirtualObjectPlaced() {
                        chooseObject(addObjectButton)
                        return
                }

                currentGesture = currentGesture?.updateGestureFromTouches(touches,
.touchEnded)
        }
```

```
	override func touchesCancelled(_ touches: Set<UITouch>, with event: UIEvent?)
{
		if !VirtualObjectsManager.shared.isAVirtualObjectPlaced() {
			return
		}
		currentGesture = currentGesture?.updateGestureFromTouches(touches,
.touchCancelled)
	}
}

// MARK: - UIPopoverPresentationControllerDelegate
extension MainViewController: UIPopoverPresentationControllerDelegate {
	func adaptivePresentationStyle(for controller: UIPresentationController) ->
UIModalPresentationStyle {
		return .none
	}

	func popoverPresentationControllerDidDismissPopover(_
popoverPresentationController: UIPopoverPresentationController) {
		updateSettings()
	}
}

// MARK: - VirtualObjectSelectionViewControllerDelegate
extension MainViewController: VirtualObjectSelectionViewControllerDelegate {
	func virtualObjectSelectionViewController(_:
VirtualObjectSelectionViewController, object: VirtualObject) {
		loadVirtualObject(object: object)
	}

	func loadVirtualObject(object: VirtualObject) {
		// Show progress indicator
		let spinner = UIActivityIndicatorView()
		spinner.center = addObjectButton.center
		spinner.bounds.size = CGSize(width: addObjectButton.bounds.width - 5,
height: addObjectButton.bounds.height - 5)
		addObjectButton.setImage(#imageLiteral(resourceName: "buttonring"),
for: [])
		sceneView.addSubview(spinner)
		spinner.startAnimating()

		DispatchQueue.global().async {
			self.isLoadingObject = true
			object.viewController = self
			VirtualObjectsManager.shared.addVirtualObject(virtualObject:
object)
```

```
        VirtualObjectsManager.shared.setVirtualObjectSelected(virtualObject: object)

                        object.loadModel()

                        DispatchQueue.main.async {
                                if let lastFocusSquarePos =
self.focusSquare?.lastPosition {

        self.setNewVirtualObjectPosition(lastFocusSquarePos)
                                } else {
                                        self.setNewVirtualObjectPosition(SCNVector3Zero)
                                }

                                spinner.removeFromSuperview()

                                // Update the icon of the add object button
                                let buttonImage = UIImage.composeButtonImage(from:
object.thumbImage)
                                let pressedButtonImage =
UIImage.composeButtonImage(from: object.thumbImage, alpha: 0.3)
                                self.addObjectButton.setImage(buttonImage, for: [])
                                self.addObjectButton.setImage(pressedButtonImage, for:
[.highlighted])
                                self.isLoadingObject = false
                        }
                }
        }
}

// MARK: - ARSCNViewDelegate
extension MainViewController: ARSCNViewDelegate {
        func renderer(_ renderer: SCNSceneRenderer, updateAtTime time: TimeInterval) {
                refreshFeaturePoints()

                DispatchQueue.main.async {
                        self.updateFocusSquare()
                        self.hitTestVisualization?.render()

                        // If light estimation is enabled, update the intensity of the
model's lights and the environment map
                        if let lightEstimate = self.session.currentFrame?.lightEstimate
{

        self.sceneView.enableEnvironmentMapWithIntensity(lightEstimate.ambientIntensit
y / 40)
```

```
                        } else {
                                self.sceneView.enableEnvironmentMapWithIntensity(25)
                        }
                }
        }

        func renderer(_ renderer: SCNSceneRenderer, didAdd node: SCNNode, for anchor:
ARAnchor) {
                DispatchQueue.main.async {
                        if let planeAnchor = anchor as? ARPlaneAnchor {
                                self.addPlane(node: node, anchor: planeAnchor)
                                self.checkIfObjectShouldMoveOntoPlane(anchor:
planeAnchor)
                        }
                }
        }

        func renderer(_ renderer: SCNSceneRenderer, didUpdate node: SCNNode, for
anchor: ARAnchor) {
                DispatchQueue.main.async {
                        if let planeAnchor = anchor as? ARPlaneAnchor {
                                if let plane = self.planes[planeAnchor] {
                                        plane.update(planeAnchor)
                                }
                                self.checkIfObjectShouldMoveOntoPlane(anchor:
planeAnchor)
                        }
                }
        }

        func renderer(_ renderer: SCNSceneRenderer, didRemove node: SCNNode, for
anchor: ARAnchor) {
                DispatchQueue.main.async {
                        if let planeAnchor = anchor as? ARPlaneAnchor, let plane =
self.planes.removeValue(forKey: planeAnchor) {
                                plane.removeFromParentNode()
                        }
                }
        }
}

// MARK: Virtual Object Manipulation
extension MainViewController {
        func displayVirtualObjectTransform() {
                guard let object =
VirtualObjectsManager.shared.getVirtualObjectSelected(),
```

```
                        let cameraTransform = session.currentFrame?.camera.transform
else {
                        return
                }

                // Output the current translation, rotation & scale of the virtual
object as text.
                let cameraPos = SCNVector3.positionFromTransform(cameraTransform)
                let vectorToCamera = cameraPos - object.position

                let distanceToUser = vectorToCamera.length()

                var angleDegrees = Int(((object.eulerAngles.y) * 180) / Float.pi) % 360
                if angleDegrees < 0 {
                        angleDegrees += 360
                }

                let distance = String(format: "%.2f", distanceToUser)
                let scale = String(format: "%.2f", object.scale.x)
                textManager.showDebugMessage("Distance: \(distance) m\nRotation:
\(angleDegrees)°\nScale: \(scale)x")
        }

        func moveVirtualObjectToPosition(_ pos: SCNVector3?, _ instantly: Bool, _
filterPosition: Bool) {

                guard let newPosition = pos else {
                        textManager.showMessage("CANNOT PLACE OBJECT\nTry moving left or
right.")
                        // Reset the content selection in the menu only if the content
has not yet been initially placed.
                        if !VirtualObjectsManager.shared.isAVirtualObjectPlaced() {
                                resetVirtualObject()
                        }
                        return
                }

                if instantly {
                        setNewVirtualObjectPosition(newPosition)
                } else {
                        updateVirtualObjectPosition(newPosition, filterPosition)
                }
        }

        func worldPositionFromScreenPosition(_ position: CGPoint,
                                             objectPos: SCNVector3?,
```

```
                                                infinitePlane: Bool = false) ->
(position: SCNVector3?,

                                                        planeAnchor: ARPlaneAnchor?,

                                                        hitAPlane: Bool) {

                // ------------------------------------------------------------------
-----------
                // 1. Always do a hit test against exisiting plane anchors first.
                //    (If any such anchors exist & only within their extents.)
                let planeHitTestResults = sceneView.hitTest(position, types:
.existingPlaneUsingExtent)
                if let result = planeHitTestResults.first {

                        let planeHitTestPosition =
SCNVector3.positionFromTransform(result.worldTransform)
                        let planeAnchor = result.anchor

                        // Return immediately - this is the best possible outcome.
                        return (planeHitTestPosition, planeAnchor as? ARPlaneAnchor,
true)
                }

                // ------------------------------------------------------------------
-----------
                // 2. Collect more information about the environment by hit testing
against
                //    the feature point cloud, but do not return the result yet.
                var featureHitTestPosition: SCNVector3?
                var highQualityFeatureHitTestResult = false

                let highQualityfeatureHitTestResults =
                        sceneView.hitTestWithFeatures(position,
coneOpeningAngleInDegrees: 18, minDistance: 0.2, maxDistance: 2.0)

                if !highQualityfeatureHitTestResults.isEmpty {
                        let result = highQualityfeatureHitTestResults[0]
                        featureHitTestPosition = result.position
                        highQualityFeatureHitTestResult = true
                }

                // ------------------------------------------------------------------
-----------
                // 3. If desired or necessary (no good feature hit test result): Hit
test
```

```
                //    against an infinite, horizontal plane (ignoring the real world).
                if (infinitePlane && dragOnInfinitePlanesEnabled) ||
!highQualityFeatureHitTestResult {

                        let pointOnPlane = objectPos ?? SCNVector3Zero

                        let pointOnInfinitePlane =
sceneView.hitTestWithInfiniteHorizontalPlane(position, pointOnPlane)
                        if pointOnInfinitePlane != nil {
                                return (pointOnInfinitePlane, nil, true)
                        }
                }

                // ------------------------------------------------------------------
-----------
                // 4. If available, return the result of the hit test against high
quality
                //    features if the hit tests against infinite planes were skipped or
no
                //    infinite plane was hit.
                if highQualityFeatureHitTestResult {
                        return (featureHitTestPosition, nil, false)
                }

                // ------------------------------------------------------------------
-----------
                // 5. As a last resort, perform a second, unfiltered hit test against
features.
                //    If there are no features in the scene, the result returned here
will be nil.
                let unfilteredFeatureHitTestResults =
sceneView.hitTestWithFeatures(position)
                if !unfilteredFeatureHitTestResults.isEmpty {
                        let result = unfilteredFeatureHitTestResults[0]
                        return (result.position, nil, false)
                }

                return (nil, nil, false)
        }

        func setNewVirtualObjectPosition(_ pos: SCNVector3) {

                guard let object =
VirtualObjectsManager.shared.getVirtualObjectSelected(),
                        let cameraTransform = session.currentFrame?.camera.transform
else {
```

```
                        return
                }

                recentVirtualObjectDistances.removeAll()

                let cameraWorldPos = SCNVector3.positionFromTransform(cameraTransform)
                var cameraToPosition = pos - cameraWorldPos
                cameraToPosition.setMaximumLength(DEFAULT_DISTANCE_CAMERA_TO_OBJECTS)

                object.position = cameraWorldPos + cameraToPosition

                if object.parent == nil {
                        sceneView.scene.rootNode.addChildNode(object)
                }
        }

        func resetVirtualObject() {
                VirtualObjectsManager.shared.resetVirtualObjects()

                addObjectButton.setImage(#imageLiteral(resourceName: "add"), for: [])
                addObjectButton.setImage(#imageLiteral(resourceName: "addPressed"),
for: [.highlighted])
        }

        func updateVirtualObjectPosition(_ pos: SCNVector3, _ filterPosition: Bool) {
                guard let object =
VirtualObjectsManager.shared.getVirtualObjectSelected() else {
                        return
                }

                guard let cameraTransform = session.currentFrame?.camera.transform else
{
                        return
                }

                let cameraWorldPos = SCNVector3.positionFromTransform(cameraTransform)
                var cameraToPosition = pos - cameraWorldPos
                cameraToPosition.setMaximumLength(DEFAULT_DISTANCE_CAMERA_TO_OBJECTS)

                // Compute the average distance of the object from the camera over the
last ten
                // updates. If filterPosition is true, compute a new position for the
object
                // with this average. Notice that the distance is applied to the vector
from
```

```
                // the camera to the content, so it only affects the percieved distance
of the
                // object - the averaging does _not_ make the content "lag".
                let hitTestResultDistance = CGFloat(cameraToPosition.length())

                recentVirtualObjectDistances.append(hitTestResultDistance)
                recentVirtualObjectDistances.keepLast(10)

                if filterPosition {
                        let averageDistance = recentVirtualObjectDistances.average!

                        cameraToPosition.setLength(Float(averageDistance))
                        let averagedDistancePos = cameraWorldPos + cameraToPosition

                        object.position = averagedDistancePos
                } else {
                        object.position = cameraWorldPos + cameraToPosition
                }
        }

        func checkIfObjectShouldMoveOntoPlane(anchor: ARPlaneAnchor) {
                guard let object =
VirtualObjectsManager.shared.getVirtualObjectSelected(),
                        let planeAnchorNode = sceneView.node(for: anchor) else {
                        return
                }

                // Get the object's position in the plane's coordinate system.
                let objectPos = planeAnchorNode.convertPosition(object.position, from:
object.parent)

                if objectPos.y == 0 {
                        return; // The object is already on the plane
                }

                // Add 10% tolerance to the corners of the plane.
                let tolerance: Float = 0.1

                let minX: Float = anchor.center.x - anchor.extent.x / 2 -
anchor.extent.x * tolerance
                let maxX: Float = anchor.center.x + anchor.extent.x / 2 +
anchor.extent.x * tolerance
                let minZ: Float = anchor.center.z - anchor.extent.z / 2 -
anchor.extent.z * tolerance
                let maxZ: Float = anchor.center.z + anchor.extent.z / 2 +
anchor.extent.z * tolerance
```

```
                if objectPos.x < minX || objectPos.x > maxX || objectPos.z < minZ ||
objectPos.z > maxZ {
                        return
                }

                // Drop the object onto the plane if it is near it.
                let verticalAllowance: Float = 0.03
                if objectPos.y > -verticalAllowance && objectPos.y < verticalAllowance
{
                        textManager.showDebugMessage("OBJECT MOVED\nSurface detected
nearby")

                        SCNTransaction.begin()
                        SCNTransaction.animationDuration = 0.5
                        SCNTransaction.animationTimingFunction =
CAMediaTimingFunction(name: kCAMediaTimingFunctionEaseInEaseOut)
                        object.position.y = anchor.transform.columns.3.y
                        SCNTransaction.commit()
                }
        }
}
```

Додаток В. Критерії оцінювання комерційного потенціалу

Таблиця 2.1 – Рекомендовані критерії оцінювання комерційного потенціалу розробки та їх можлива бальна оцінка

Критерії оцінювання та бали (за 5-ти бальною шкалою)					
Критерій	0	1	2	3	4
Технічна здійсненність концепції:					
1	Достовірність концепції не підтверджена	Концепція підтверджена експертними висновками	Концепція підтверджена розрахунками	Концепція перевірена на практиці	Перевірено роботоздатність продукту в реальних умовах
Ринкові переваги (недоліки):					
2	Багато аналогів на малому ринку	Мало аналогів на малому ринку	Кілька аналогів на великому ринку	Один аналог на великому ринку	Продукт не має аналогів на великому ринку
3	Ціна продукту значно вища за ціни аналогів	Ціна продукту дещо вища за ціни аналогів	Ціна продукту приблизно дорівнює цінам аналогів	Ціна продукту дещо нижче за ціни аналогів	Ціна продукту значно нижче за ціни аналогів
4	Технічні та споживчі властивості продукту значно гірші, ніж в аналогів	Технічні та споживчі властивості продукту трохи гірші, ніж в аналогів	Технічні та споживчі властивості продукту на рівні аналогів	Технічні та споживчі властивості продукту трохи кращі, ніж в аналогів	Технічні та споживчі властивості продукту значно кращі, ніж в аналогів
5	Експлуатаційні витрати значно вищі, ніж в аналогів	Експлуатаційні витрати дещо вищі, ніж в аналогів	Експлуатаційні витрати на рівні експлуатаційних витрат аналогів	Експлуатаційні витрати трохи нижчі, ніж в аналогів	Експлуатаційні витрати значно нижчі, ніж в аналогів

Продовження таблиці 2.1

Критерії оцінювання та бали (за 5-ти бальною шкалою)					
Крите.	0	1	2	3	4
Ринкові перспективи					
6	Ринок малий і не має позитивної динаміки	Ринок малий, але має позитивну динаміку	Середній ринок з позитивною динамікою	Великий стабільний ринок	Великий ринок з позитивною динамікою
7	Активна конкуренція великих компаній на ринку	Активна конкуренція	Помірна конкуренція	Незначна конкуренція	Конкурентів немає
Практична здійсненність					
8	Відсутні фахівці як з технічної, так і з комерційної реалізації ідеї	Необхідно наймати фахівців або витрачати значні кошти та час на навчання наявних фахівців	Необхідне незначне навчання фахівців та збільшення їх штату	Необхідне незначне навчання фахівців	Є фахівці з питань як з технічної, так і з комерційної реалізації ідеї
9	Потрібні значні фінансові ресурси, які відсутні. Джерела фінансування ідеї відсутні	Потрібні незначні фінансові ресурси. Джерела фінансування відсутні	Потрібні значні фінансові ресурси. Джерела фінансування є	Потрібні незначні фінансові ресурси. Джерела фінансування є	Не потребує додаткового фінансування
10	Необхідна розробка нових матеріалів	Потрібні матеріали, що використовуються у військово-промисловому комплексі	Потрібні дорогі матеріали	Потрібні досяжні та дешеві матеріали	Всі матеріали для реалізації ідеї відомі та давно використовуються у виробництві
11	Термін реалізації ідеї більший за 10 років	Термін реалізації ідеї більший за 5 років. Термін окупності інвестицій більше 10-ти років	Термін реалізації ідеї від 3-х до 5-ти років. Термін окупності інвестицій більше 5-ти років	Термін реалізації ідеї менше 3-х років. Термін окупності інвестицій від 3-х до 5-ти років	Термін реалізації ідеї менше 3-х років. Термін окупності інвестицій менше 3-х років
12	Необхідна розробка регламентних документів та отримання великої кількості дозвільних документів на виробництво та реалізацію продукту	Необхідно отримання великої кількості дозвільних документів на виробництво та реалізацію продукту, що вимагає значних коштів та часу	Процедура отримання дозвільних документів для виробництва та реалізації продукту вимагає незначних коштів та часу	Необхідно тільки повідомлення відповідним органам про виробництво та реалізацію продукту	Відсутні будь-які регламентні обмеження на виробництво та реалізацію продукту

Додаток Г. Ілюстративний матеріал

ІЛЮСТРАТИВНИЙ МАТЕРІАЛ ДО ЗАХИСТУ МАГІСТЕРСЬКОЇ КВАЛІФІКАЦІЙНОЇ РОБОТИ

Завідувач кафедри ПЗ, д.т.н., професор _____ О. Н. Романюк

Науковий керівник, к.т.н., доцент _____ Д. І. Катєльніков

Рецензент, к.т.н., доцент _____ І. Р. Арсенюк

Нормоконтроль, к.т.н., доцент _____ Д. І. Катєльніков

Виконавець, студент групи 2ПІ-18м _____ В. В. Туйчев

Розробка методів і програмного забезпечення доповненої реальності для розпізнавання рухів з використанням технологій Swift, ARKit, CoreML

Автор: ст. гр. 2ПІ-18м Туйчев В.В

Науковий керівник: к.т.н., доцент Катєльніков Д.І.

Слайд 1 – Тема, автор, науковий керівник

Актуальність

- Багато підходів було зроблено за допомогою камер та алгоритмів комп'ютерного зору для інтерпретації мови жестів. Використовуючи концепцію розпізнавання жестів, можна керувати пристроєм лише вказівним пальцем. Це допоможе максимально зменшити введення даних за механічним способом.

Слайд 2 – Актуальність

Мета, об’єкт та предмет дослідження

- **Мета** – покращення взаємодії користувач-комп’ютер за рахунок поєднання методів створення доповненої реальності і розпізнавання рухів.
- **Об’єкт дослідження** – процес інтерактивної взаємодії користувач-комп’ютер.
- **Предмет дослідження** – методи та засоби розпізнавання жестів рук.

Слайд 3 – Мета, об’єкт та предмет дослідження

Задачі

- провести аналіз існуючих методів і засобів створення доповненої реальності для визначення напрямків їх ефективного використання з системою розпізнавання рухів;
- запропонувати нові:
  a) методи інтерактивної взаємодії з урахуванням розпізнавання жестів рук, положення долоні, кутів суглобів тощо;
  b) методи поєднання доповненої реальності і розпізнавання рухів;
- розробити програмні компоненти та систему доповненої реальності на основі запропонованих методів;
- провести експериментальні дослідження розроблених засобів інтерактивної взаємодії користувач-комп’ютер на основі доповненої реальності та розпізнавання жестів рук, положення долоні.

Слайд 4 – Задачі

Наукова новизна

- **Наукова новизна отриманих результатів.**
- 1. Подальшого розвитку отримав метод інтерактивної взаємодії користувач-комп'ютер у якому, на відміну від існуючих, використано не тільки дані маніпуляторів миша та клавіатура, але й враховується результати розпізнавання жестів рук, положення долоні, кутів суглобів, що дозволяє значно розширити діапазон можливих команд.
- 2. Подальшого розвитку отримав метод формування доповненої реальності у якому, на відміну від існуючих, використано нейронну мережу, що дозволяє урізноманітнити команди за рахунок взаємодії з віртуальними об'єктами в реальному світі.

Слайд 5 – Наукова новизна

Практичне значення

- Практична цінність одержаних результатів полягає в тому, що на основі отриманих в магістерській кваліфікаційній роботі теоретичних положень запропоновано алгоритми та розроблено програмний засіб для взаємодії користувач-комп'ютер на основі поєднання методів створення доповненої реальності і розпізнавання рухів для інтерактивних інформаційних систем.

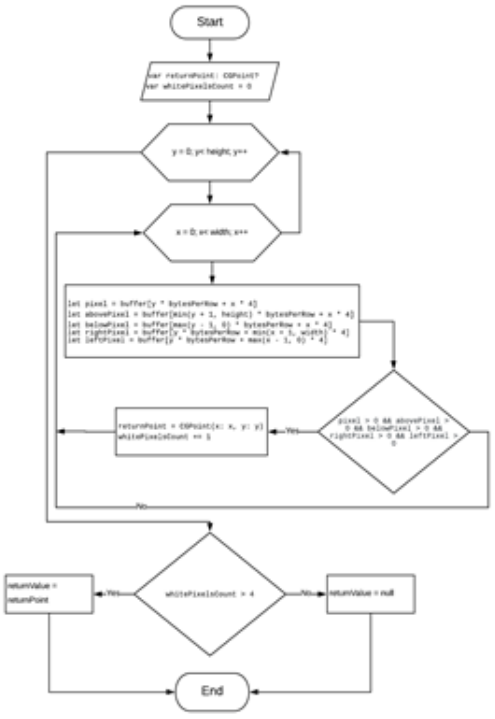
Слайд 6 – Практичне значення

Порівняння з аналогами

Критерій	ARCore Elements	VR Gesture Player	Wanna Nails	Face Racer – no hands!	Hands Gesture AR
Розпізнавання рухів	-	+	+	+	+
Використання AR	+	-	+	-	+
Взаємодія з віртуальними об'єктами	+	+	-	+	+

Слайд 7 – Порівняння з аналогами

Алгоритм знаходження найвищої точки

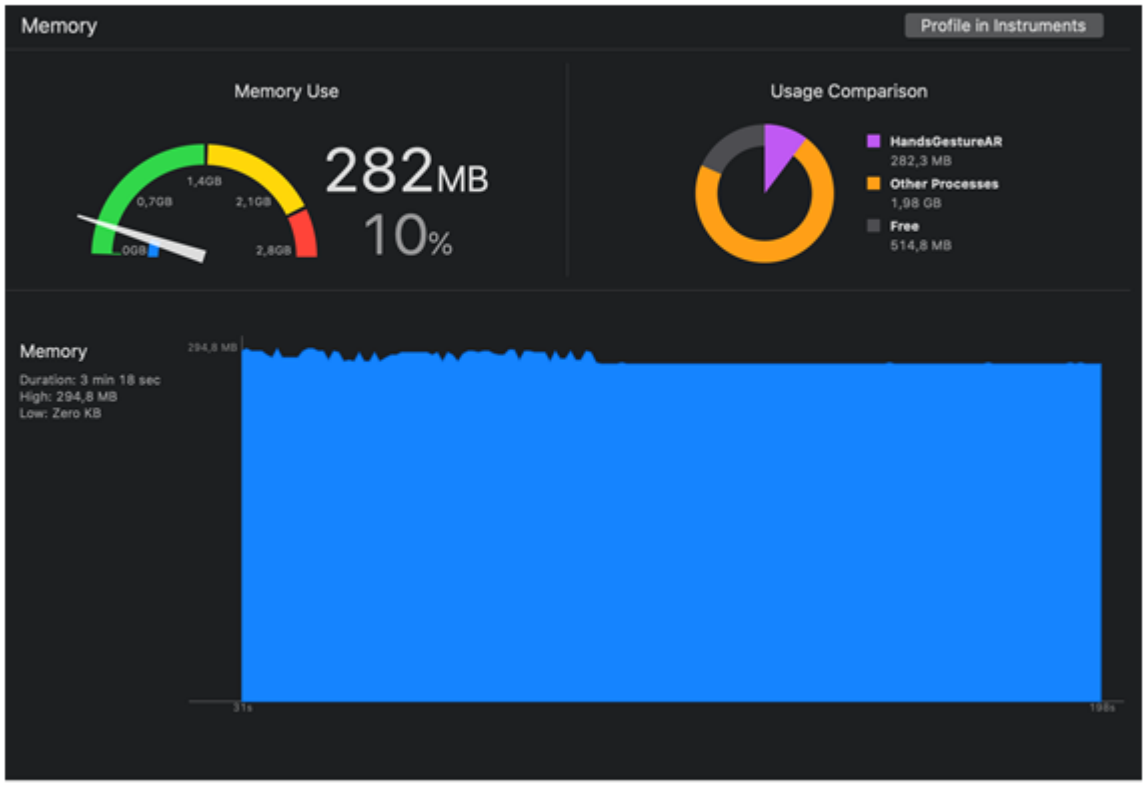


Слайд 8 – Алгоритм знаходження найвищої точки

Слайд 9 – Технології

Слайд 10 – Тестування програми

Ресурсозатратність



Слайд 11 – Ресурсозатратність додатку

Апробація матеріалів магістерської кваліфікаційної роботи

- Основні положення магістерської кваліфікаційної роботи доповідалися та обговорювалися на II Всеукраїнській науково-практичній інтернет-конференції здобувачів вищої освіти і молодих учених "Інформаційно-комп'ютерні технології: стан, досягнення та перспективи розвитку" (м. Житомир, 2019) та II Всеукраїнській науково-технічній конференції «Комп’ютерні технології: інновації, проблеми, рішення» (м. Житомир, 2019).

Слайд 12 – Апробація матеріалів