

Міністерство освіти і науки України
Національний університет «Львівська політехніка»
Інститут прикладної математики та фундаментальних наук

Кафедра прикладної математики

Звіт

про проходження практики
за темою кваліфікаційної роботи
«Розроблення мобільного додатку для визначення хмарності неба»

Виконав:

студент групи ПМ-41

Тимощук П. Т.

Керівник роботи:

Топилко П. І.

(оцінка)

(підпис керівника кваліф. роботи)

Керівник практики:

Антонова Т. М.

(дата)

(підпис керівника практики)

Львів 2021

Міністерство освіти і науки України
Національний університет «Львівська політехніка»

Завдання та результати проходження практики

Студент Тимочук Павло Тарасович
(прізвище, ім'я, по батькові)

Освітній рівень бакалавр спеціальність прикладна математика

Скерований на практику за темою БКР
(вид практики)

в місто Львів на ТзОВ «Мала комп'ютерна академія ШІАГ Львів3»
(назва підприємства, організації, установи)

згідно договору № 12-10-2021 від 6.04.21р.

Термін практики: від 03.05.2021р. до 15.05.2021р.
(з врахуванням проїзду туди і назад)

Керівник практики від університету доцент каф. ПМ Манзій О.С.
(посада, прізвище, ім'я, по батькові та підпис)



/Директор інституту ІМФН
(абрєвіатура)

[Signature]
(підпис)

« » 20 р

Відмітка про проходження практики:

Прибув на базу практики „03” травня 2021 року
Мельниченко О.П. /керівник навчальної частини/
(підпис) (посада, прізвище та ініціали відповідальної особи)

Печатка бази практики

Вийшов з бази практики „15” травня 2021 року
Мельниченко О.П. /керівник навчальної частини/
(підпис) (посада, прізвище та ініціали відповідальної особи)

Печатка бази практики

Зміст завдання

(перелік питань, які підлягають виконанню)
(заповнює керівник БКР)

Завдання видав: _____

(посада, прізвище, ініціали керівника БКР, підпис і дата)

Завдання отримав: _____

(прізвище, ініціали, підпис і дата)

Відгук та оцінка роботи студента на практиці

(оцінка керівника БКР виконання завдання)
(заповнює керівник БКР)

(посада, прізвище, ім'я, по батькові та підпис керівника БКР)

«__» __20__р.

Відгук керівника практики від кафедри про виконання завдання

Дата складання заліку «__» _____20__р.

Оцінка:

За національною шкалою

(прописом)

Кількість балів

(цифрами і прописом)

Члени комісії, які приймали залік:

(посада, прізвище, ім'я, по батькові та підписи)

Зміст

Вступ.....	6
1. Уточнення виявлення пікселів хмари на основі середнього кольору зображення.....	7
2. Підвищення швидкодії програми.....	9
3. Зміна демонстрації та заокруглення значення відсотка хмарності.....	11
Опис програми.....	12
Висновки.....	13
Список використаної літератури.....	14
Додаток 1.....	15
Додаток 2.....	24






Розглянуто задачу уточнення виявлення пікселів хмари на основі середнього кольору, підвищення швидкодії програми та зміни демонстрації і заокруглення значення відсотка хмарності. Написано програму, яка реалізовує дану задачу. У ході роботи виконано порівняння уточнень та підвищення швидкодії програми.

The problem of refining the detection of cloud pixels based on the average color, increasing the speed of the program and changing the demonstration and rounding of the cloud percentage is considered. A program has been written that implements this task. In the course of work performed comparison of clarification and time of calculation.

Вступ

Метод оцінки характеристик пікселя для визначення його належності до хмари є робочим, але для різних зображень їхні середні кольори можуть сильно відрізнятись, що призведе до значної втрати точності. Також швидкодія даного методу є низькою, оскільки обчислюється та порівнюється значення кожного пікселя зображення.

1. Уточнення виявлення пікселів хмари на основі середнього кольору зображення

Зображення	Початковий результат	Ітерації	Проміжний результат	Очікуваний результат
	61% Середній колір зображення: $r = 0.822368$ $g = 0.842105$ $b = 0.888158$	1 ітерація	72%	80-90%
		2 ітерація	70%	
		3 ітерація	73%	
		4 ітерація	88%	
		5 ітерація	80%	
	68% Середній колір зображення: $r = 0.521472$ $g = 0.662577$ $b = 0.846626$	1 ітерація	11%	40-50%
		2 ітерація	2%	
		3 ітерація	15%	
		4 ітерація	100%	
		5 ітерація	90%	
	12% Середній колір зображення: $r = 0.247934$ $g = 0.586777$ $b = 0.867769$	1 ітерація	0%	0-5%
		2 ітерація	0%	
		3 ітерація	0%	
		4 ітерація	0%	
		5 ітерація	0%	
	45% Середній колір зображення: $r = 0.598639$ $g = 0.755102$ $b = 0.85034$	1 ітерація	45%	40-50%
		2 ітерація	42%	
		3 ітерація	46%	
		4 ітерація	63%	
		5 ітерація	51%	
	55% Середній колір зображення: $r = 0.623762$ $g = 0.69802$ $b = 0.80198$	1 ітерація	54%	70-80%
		2 ітерація	48%	
		3 ітерація	55%	
		4 ітерація	85%	
		5 ітерація	75%	

Зміна результату в залежності від корегування середнього кольору

Ітерація	Корегований середній колір
1 ітерація	(r = 0.6; g = 0.75 ; b = 0.85)
2 ітерація	(r = 0.6; g = 0.5 ; b = 0.85)
3 ітерація	(r = 0.6 ; g = 0.85 ; b = 0.85)
4 ітерація	(r = 0.4 ; g = 0.85; b = 0.85)
5 ітерація	(r = 0.5 ; g = 0.85; b = 0.8)

Зміна корегованого середнього кольору в залежності від попередньо отриманих результатів

Проаналізувавши дані з таблиці «Зміна корегованого середнього кольору в залежності від попередньо отриманих результатів» я дійшов висновку, що оптимальним корегованим середнім кольором буде (**r = 0.5**; **g = 0.85**; **b = 0.8**).

2. Підвищення швидкодії програми

Зображення і його роздільна здатність	Ітерації	Час обчислень	Результат обчислень
w: 4752 h: 3168	1 ітерація	113 с	81.3817%
	2 ітерація	30 с	81.381714%
	3 ітерація	13 с	81.40434%
	4 ітерація	10 с	81.29915%
	5 ітерація	9 с	81.342735%
w: 4000 h: 3000	1 ітерація	59 с	42.43563%
	2 ітерація	18 с	42.435634%
	3 ітерація	9 с	42.41984%
	4 ітерація	8 с	42.41055%
	5 ітерація	7 с	42.42921%
w: 4000 h: 3000	1 ітерація	45 с	0.07061231%
	2 ітерація	15 с	0.0702467%
	3 ітерація	9 с	0.0664146%
	4 ітерація	7 с	0.050503287%
	5 ітерація	7 с	0.034644034%
w: 4000 h: 3000	1 ітерація	89 с	52.706326%
	2 ітерація	24 с	52.70325%
	3 ітерація	10 с	52.682503%
	4 ітерація	8 с	52.8075%
	5 ітерація	7 с	53.95%
w: 4000 h: 3000	1 ітерація	71 с	74.36131%
	2 ітерація	20 с	74.36032%
	3 ітерація	9 с	74.37062%
	4 ітерація	8 с	74.45019%
	5 ітерація	7 с	73.603035%

Зміна часу та результату обчислення хмарності в залежності від зміни розмірів груп об'єднання пікселів

Використовуючи дані з таблиці «Зміна часу та результату обчислення хмарності в залежності від зміни розмірів груп об'єднання пікселів» побудую наступну таблицю:

Ітерація	Розмір груп об'єднання пікселів	Середній приріст швидкодії*	Середнє падіння точості**
1 ітерація	1*1	1x	0%
2 ітерація	5*5	3.46x	0,1049929976%
3 ітерація	30*30	2.12x	1,213495323%
4 ітерація	100*100	1.22x	5,790017853%
5 ітерація	500*500	1.11x	10,87600793%

Середній приріст швидкодії та падіння точості при зміні розмірів груп об'єднання пікселів

* Середній приріст швидкодії відносно попередньої ітерації

** Середнє падіння точості відносно початкового результату

Проаналізувавши дані з таблиці «Середній приріст швидкодії та падіння точості при зміні розмірів груп об'єднання пікселів» я дійшов висновку що оптимальним буде розбиття на групи 30*30 пікселів. У такому випадку час обчислення буде меншим в 7.35 разів у порівнянні з часом обчислення без розбиття на групи, а похибки будуть достатньо малими щоб ними знехтувати. Також слід зазначити, що для зображень з низькою роздільною здатністю великі розміри груп розбиття приведуть до значної втрати точності обчислень, тому для зображення необхідно корегувати розмір розбиття залежно від роздільної здатності зображення.

3. Зміна демонстрації та заокруглення значення відсотка хмарності

Зміна демонстрації відсотка хмарності:

```
func addCloudPercentageLabel(view: UIView) {
    cloudPercentageLabel.frame.size = CGSize(width: view.frame.width,
height: 50)
    cloudPercentageLabel.frame.origin = CGPoint(x: 0, y:
view.frame.height/2)
    view.addSubview(cloudPercentageLabel)
    cloudPercentageLabel.textAlignment = .center
    cloudPercentageLabel.font = UIFont(name: "AppleSDGothicNeo-Bold",
size: 30)
}
```

Заокруглення значення відсотка хмарності:

```
extension Float {
    func roundedValue(base: Int) -> String {
        let roundedInt = Int(roundf(self))%base < (base/2)+1 ? Int(roundf(self))/
base*base : (Int(roundf(self))/base+1)*base
        return String(roundedInt)+"%"
    }
}
```

Опис програми

Програма написана на мові програмування Swift у середовищі Xcode. Вона реалізовує задачу уточнення виявлення пікселів хмари на основі середнього кольору, підвищення швидкодії програми та зміни демонстрації і заокруглення значення відсотка хмарності.

Вхідні дані програми:

Зображення з галереї девайсу

У програмі реалізовано такі головні функції:

1. `imagePickerController` (отримання зображення з галереї);
2. `addPhotoAndCalculateCloudPercentage` (перетворення зображення та обчислення відсотку хмарності);
3. `calculateVSC` (класифікація окремі пікселі на сегменти неба та хмари на основі кольору пікселя);
4. `averageColor` (отримання середнього кольору зображення);
5. `correctionValue` (корегування параметру в залежності від середнього кольору)
6. `roundedValue` (округлення числа)

У результаті виконання програми отримуємо такі вихідні дані:

1. Відсоток хмарності неба, у випадку якщо вхідне зображення містить фрагмент неба;
2. Повідомлення про те, що вхідне зображення не містить фрагментів неба.

Результат програми виводиться на екран.

Висновки

У ході роботи розглянуто задачу уточнення виявлення пікселів хмари на основі середнього кольору, підвищення швидкодії програми та зміни демонстрації і заокруглення значення відсотка хмарності. Написано програму, яка реалізовує дану задачу. У ході роботи виконано порівняння уточнень та підвищення швидкодії програми.

У результаті можна зробити висновок, що для забезпечення оптимальної швидкодії необхідно використовувати розбиття на групи 30×30 пікселів, а для зображень з низькою роздільною здатністю необхідно корегувати розмір розбиття залежно від роздільної здатності зображення.

Для уточнення виявлення пікселів хмари оптимальним середнім кольором буде ($r = 0.5$; $g = 0.85$; $b = 0.8$).

Список використаної літератури

1. Cloud cover. https://en.wikipedia.org/wiki/Cloud_cover
2. Deep Convolutional Neural Network for Cloud Coverage Estimation from Snapshot Camera Images. https://www.researchgate.net/publication/321984089_Deep_Convolutional_Neural_Network_for_Cloud_Coverage_Estimation_from_Snapshot_Camera_Images
3. Image Segmentation. <https://www.fritz.ai/features/image-segmentation.html>
4. Image Segmentation iOS. <https://docs.fritz.ai/develop/vision/image-segmentation/#ios>
5. SwiftUI: Read the Average Color of an Image. <https://medium.com/swlh/swiftui-read-the-average-color-of-an-image-c736adb43000>
6. Сегментація зображення. https://uk.wikipedia.org/wiki/Сегментація_зображення

Додаток 1

Текст програми

```
// ViewController.swift
// Cloudy
// Created by Павло Тимощук on 25.11.2020.

import UIKit
import Photos

class ViewController: UIViewController, UIImagePickerControllerDelegate,
UINavigationControllerDelegate {
    var imageView: UIImageView!
    var imagePicker = UIImagePickerController()
    var isCurrentImageSky = Bool()
    var image = UIImage()
    var skyImage = UIImage()
    var cloudPercentageLabel = UILabel()

    override func viewDidLoad() {
        super.viewDidLoad()
        addBackground(view: self.view)
        addSelectImageButton(view: self.view)
        addCloudPercentageLabel(view: self.view)
        self.view.insertSubview(imageViewCreating(), at: 0)
    }
    func addBackground(view: UIView) {
        let layer0 = CAGradientLayer()
        layer0.colors = [
            UIColor(red: 0.883, green: 0.968, blue: 1, alpha: 1).CGColor,
            UIColor(red: 0.351, green: 0.816, blue: 0.992, alpha: 1).CGColor,
            UIColor(red: 0.197, green: 0.269, blue: 0.296, alpha: 1).CGColor
        ]
        layer0.locations = [0, 0.5, 1]
        layer0.startPoint = CGPoint(x: 0.25, y: 0.5)
        layer0.endPoint = CGPoint(x: 0.75, y: 0.5)
        layer0.transform = CATransform3DMakeAffineTransform(CGAffineTransform(a: 0,
b: 1, c: -1, d: 0, tx: 1, ty: 0))
        layer0.bounds = view.bounds.insetBy(dx: -1*view.bounds.size.width, dy:
-1*view.bounds.size.height)
        layer0.position = view.center
        view.layer.addSublayer(layer0)

        let imageSize: CGFloat = 150
```



```

    let padding = (view.frame.height - 3*imageSize)/6
    let sunImage = UIImageView()
    sunImage.layer.opacity = 0.4
    sunImage.frame = CGRect(x: (view.frame.width-imageSize)/2,
                             y: padding,
                             width: imageSize,
                             height: imageSize)
    sunImage.image = UIImage(named: "sun-image")
    view.addSubview(sunImage)
    let cloudedSun = UIImageView()
    cloudedSun.layer.opacity = 0.4
    cloudedSun.frame = CGRect(x: (view.frame.width-imageSize)/2,
                              y: (view.frame.height-imageSize)/2,
                              width: imageSize,
                              height: imageSize)
    cloudedSun.image = UIImage(named: "clouded-sun-image")
    view.addSubview(cloudedSun)
    let cloud = UIImageView()
    cloud.layer.opacity = 0.4
    cloud.frame = CGRect(x: (view.frame.width-imageSize)/2,
                          y: view.frame.height-imageSize-padding,
                          width: imageSize,
                          height: imageSize)
    cloud.image = UIImage(named: "cloud-image")
    view.addSubview(cloud)
}
func addSelectImageButton(view: UIView) {
    let selectImageButton = UIButton()
    let selectImageButtonSize = CGSize(width: 230, height: 60)
    selectImageButton.frame = CGRect(x: (view.frame.width-
selectImageButtonSize.width)/2,
                                     y: view.frame.height/2 + 115,
                                     width: selectImageButtonSize.width,
                                     height: selectImageButtonSize.height)
    selectImageButton.addTarget(self, action: #selector(selectImageButtonAction),
for: .touchUpInside)
    let shadows = UIView()
    shadows.isUserInteractionEnabled = false
    shadows.frame = CGRect(x: 0,
                           y: 0,
                           width: selectImageButtonSize.width,
                           height: selectImageButtonSize.height)
    shadows.clipsToBounds = false
    selectImageButton.addSubview(shadows)
    let shadowPath0 = UIBezierPath(roundedRect: shadows.bounds, cornerRadius: 20)
    let layer0 = CALayer()

```

```

layer0.shadowPath = shadowPath0.cgPath
layer0.shadowColor = UIColor(red: 0, green: 0, blue: 0, alpha: 0.2).cgColor
layer0.shadowOpacity = 1
layer0.shadowRadius = 3
layer0.shadowOffset = CGSize(width: 4, height: 4)
layer0.bounds = shadows.bounds
layer0.position = shadows.center
shadows.layer.addSublayer(layer0)
let layer1 = CALayer()
layer1.bounds = shadows.bounds
layer1.position = shadows.center
layer1.cornerRadius = 20
layer1.borderWidth = 3
layer1.borderColor = UIColor(red: 0.355, green: 0.717, blue: 0.979, alpha: 1).cgColor
layer1.backgroundColor = UIColor(red: 0.302, green: 0.635, blue: 0.765, alpha:
1).cgColor
shadows.layer.addSublayer(layer1)
let selectImageButtonText = UILabel()
selectImageButtonText.frame = CGRect(x: 0, y: 0, width: 230, height: 60)
selectImageButtonText.textColor = UIColor(red: 1, green: 1, blue: 1, alpha: 1)
selectImageButtonText.font = UIFont(name: "Roboto-Regular", size: 30)
selectImageButtonText.text = "Select image"
selectImageButton.addSubview(selectImageButtonText)
selectImageButtonText.textAlignment = .center
view.addSubview(selectImageButton)
}
func addCloudPercentageLabel(view: UIView) {
    cloudPercentageLabel.frame.size = CGSize(width: view.frame.width, height: 50)
    cloudPercentageLabel.frame.origin = CGPoint(x: 0, y: view.frame.height/2)
    view.addSubview(cloudPercentageLabel)
    cloudPercentageLabel.textAlignment = .center
    cloudPercentageLabel.font = UIFont(name: "AppleSDGothicNeo-Bold", size: 30)
}
@objc func selectImageButtonAction() {
    if UIImagePickerController.isSourceTypeAvailable(.savedPhotosAlbum){
        imagePicker.delegate = self
        imagePicker.sourceType = .savedPhotosAlbum
        imagePicker.allowsEditing = false
        present(imagePicker, animated: true, completion: nil)
    }
}
func imagePickerController(_ picker: UIImagePickerController,
                           didFinishPickingMediaWithInfo info:
[UIImagePickerController.InfoKey : Any]) {
    imagePicker.dismiss(animated: true, completion: nil)
    guard let image = info[.originalImage] as? UIImage else {

```

fatalError("Expected a dictionary containing an image, but was provided the following: \\\(info)")

```
    }
    self.imageView.image = image
    self.image = image
    self.view.bringSubviewToFront(self.imageView)
    let indicator = Indicator()
    indicator.showIndicator()
    DispatchQueue.main.asyncAfter(deadline: .now() + 0.1) {
        self.addPhotoAndCalculateCloudPercentage(image: image)
        indicator.hideIndicator()
    }
}

func addPhotoAndCalculateCloudPercentage(image: UIImage) {
    let photoAnalyzer = PhotoAnalyzer(image)
    if let image = photoAnalyzer.createSkyImage() {
        isCurrentImageSky = false
        skyImage = image
        if let cloudPercentage = photoAnalyzer.getCloudPercentage(image: image) {
            if !cloudPercentage.isNaN {
                cloudPercentageLabel.text = "Cloud Percentage " +
cloudPercentage.roundedValue(base: 5)
            } else {
                cloudPercentageLabel.text = "Sky is undetected"
            }
        } else {
            cloudPercentageLabel.text = "Sky is undetected"
        }
    } else {
        print("ERROR")
        cloudPercentageLabel.text = "Sky is undetected"
    }
}

func imageViewCreating() -> UIImageView {
    imageView = .init()
    imageView.frame.origin = CGPoint(x: 25, y: 50)
    imageView.frame.size = CGSize(width: self.view.frame.width-50, height:
self.view.frame.height/2)
    imageView.contentMode = .scaleAspectFit
    self.view.bringSubviewToFront(imageView)
    imageView.isUserInteractionEnabled = true
    let gestureRecognizer = UITapGestureRecognizer(target: self, action:
#selector(changePhoto))
    imageView.addGestureRecognizer(gestureRecognizer)
    return imageView
}
```

```

    @objc func changePhoto() {
        imageView.image = isCurrentImageSky ? image : skyImage
        isCurrentImageSky.toggle()
    }
    // MARK: - Make ALERT
    func alert(alertTitle: String, alertMessage: String, alertActionTitle: String) {
        AudioServicesPlaySystemSound(SystemSoundID(4095))
        let alert = UIAlertController(title: alertTitle, message: alertMessage,
preferredStyle: .alert)
        let action = UIAlertAction(title: alertActionTitle, style: .cancel) { (action) in }
        alert.addAction(action)
        self.present(alert, animated: true, completion: nil)
    }
}

// AnalyzingPhoto.swift
// Cloudy
// Created by Павло Тимошук on 25.11.2020.

import Foundation
import AVFoundation
import UIKit
import Vision
import Fritz
import FritzVisionSkySegmentationModelAccurate

class PhotoAnalyzer {
    init(_ photo: UIImage){
        self.photo = photo
    }
    var photo = UIImage()
    var visionModel = FritzVisionSkySegmentationModelAccurate()
    let context = CIColorContext()
    func createSkyImage() -> UIImage? {
        let fritzImage = FritzVisionImage(image: self.photo)
        guard let result = try? visionModel.predict(fritzImage),
            let mask = result.buildSingleClassMask(
                forClass: FritzVisionSkyClass.sky
            )
        else {
            return nil
        }
        guard let skyImage = createMask(of: self.photo, fromMask: mask) else { return nil }
        return skyImage
    }
    func getCloudPercentage(image: UIImage) -> Float? {

```

```

let averageColor = image.averageColor
print("Average image colors: ", averageColor!)
print("Normal colors: 0.5 0.85 0.8")
let rCorrection = Float((averageColor?.rgba.red!).correctionValue(by: 0.5))
let gCorrection = Float((averageColor?.rgba.green!).correctionValue(by: 0.85))
let bCorrection = Float((averageColor?.rgba.blue!).correctionValue(by: 0.80))
print("Color correction: ", rCorrection, gCorrection, bCorrection)
if let imageData = getDataFrom(image: image) {
    let accuracyValue = 30
    var pixelCount = 0
    var cloudPixelCount = 0
    let timeStart = Date()
    for y in 0 ..< imageData.height {
        for x in 0 ..< imageData.width {
            let index = y * imageData.width + x
            let pixel = imageData.pixels[index]
            if pixel.alpha != 0 && x%accuracyValue == y%accuracyValue {
                if calculateVSC(pixel: pixel, colorCorrection: (rCorrection, gCorrection,
bCorrection)) < 0 {
                    cloudPixelCount += 1
                }
                pixelCount += 1
            }
        }
    }
    let timeSpent = Calendar.current.dateComponents([.second], from: timeStart, to:
Date()).second!
    print("image_dimension:", "w:", imageData.width, " h:",
imageData.height, "accuracy_value: ", accuracyValue, "time_spent: ", timeSpent,
"cloud_percentage: ", Float(cloudPixelCount) / Float(pixelCount) * 100)
    return Float(cloudPixelCount) / Float(pixelCount) * 100
} else {
    return nil
}
}

func getDataFrom(image: UIImage) -> (width: Int, height: Int, pixels:
UnsafeMutableBufferPointer<Pixel>)? {
    guard let cgImage = image.cgImage else { return nil }
    let colorSpace = CGColorSpaceCreateDeviceRGB()
    var bitmapInfo: UInt32 = CGBitmapInfo.byteOrder32Big.rawValue
    bitmapInfo |= CGLImageAlphaInfo.premultipliedLast.rawValue &
CGBitmapInfo.alphaInfoMask.rawValue
    let width = Int(image.size.width)
    let height = Int(image.size.height)
    let bytesPerRow = width * 4
    let imageData = UnsafeMutablePointer<Pixel>.allocate(capacity: width * height)

```

```

guard let imageContext = CGContext(
    data: imageData,
    width: width,
    height: height,
    bitsPerComponent: 8,
    bytesPerRow: bytesPerRow,
    space: colorSpace,
    bitmapInfo: bitmapInfo
) else { return nil }
imageContext.draw(cgImage, in: CGRect(x: 0, y: 0, width: width, height: height))
return (width, height, UnsafeMutableBufferPointer<Pixel>(start: imageData, count:
width * height))
}
func calculateVSC(pixel: Pixel, colorCorrection: (r: Float, g: Float, b: Float)) -> Float {
    let r = Float(pixel.red)/255
    let g = Float(pixel.green)/255
    let b = Float(pixel.blue)/255
    let v = ([r,g,b].max()! + [r,g,b].min()!)/2
    let scyl: Float = ([r,g,b].max()! - [r,g,b].min()!) == 0 ? 0 : ([r,g,b].max()!-
[r,g,b].min()!)/(1-abs(2*v-1))
    // let scone: Float = 0
    let br = (b-r)/(b+r)
    return -6.28*r*colorCorrection.r + 0.454*g*colorCorrection.g -
4.11*b*colorCorrection.b - 1.8*scyl + 8.88*v + 1.53*br + 0.586//
*(colorCorrection.r*colorCorrection.g*colorCorrection.b)
}
func createMask(of image: UIImage, fromMask mask: UIImage, withBackground
background: UIImage? = nil) -> UIImage? {
    guard let imageCG = image.cgImage, let maskCI = mask.ciImage else { return nil }
    let imageCI = CIImage(cgImage: imageCG)
    let background = background?.cgImage != nil ? CIImage(cgImage:
background!.cgImage!) : CIImage.empty()
    guard let filter = CIFilter(name: "CIBlendWithAlphaMask") else { return nil }
    filter.setValue(imageCI, forKey: "inputImage")
    filter.setValue(maskCI, forKey: "inputMaskImage")
    filter.setValue(background, forKey: "inputBackgroundImage")
    guard let maskedImage = context.createCGImage(filter.outputImage!, from:
maskCI.extent) else {
        return nil
    }
    return UIImage(cgImage: maskedImage)
}
struct Pixel {
    public var value: UInt32
    public var red: UInt8 {
        get {

```

```

        return UInt8(value & 0xFF)
    } set {
        value = UInt32(newValue) | (value & 0xFFFFF00)
    }
}
public var green: UInt8 {
    get {
        return UInt8((value >> 8) & 0xFF)
    } set {
        value = (UInt32(newValue) << 8) | (value & 0xFFFF00FF)
    }
}
public var blue: UInt8 {
    get {
        return UInt8((value >> 16) & 0xFF)
    } set {
        value = (UInt32(newValue) << 16) | (value & 0xFF00FFFF)
    }
}
public var alpha: UInt8 {
    get {
        return UInt8((value >> 24) & 0xFF)
    } set {
        value = (UInt32(newValue) << 24) | (value & 0x00FFFFFF)
    }
}
}
}
extension UIImage {
    var averageColor: UIColor? {
        var bitmap = [UInt8](repeating: 0, count: 4)
        if #available(iOS 9.0, *) {
            let context = CIContext()
            let inputImage: CIImage = ciImage ?? CoreImage.CIImage(cgImage: cgImage!)
            let extent = inputImage.extent
            let inputExtent = CIVector(x: extent.origin.x, y: extent.origin.y, z: extent.size.width,
w: extent.size.height)
            let filter = CIFilter(name: "CIAreaAverage", parameters: [kCIInputImageKey:
inputImage, kCIInputExtentKey: inputExtent])!
            let outputImage = filter.outputImage!
            let outputExtent = outputImage.extent
            assert(outputExtent.size.width == 1 && outputExtent.size.height == 1)
            context.render(outputImage, toBitmap: &bitmap, rowBytes: 4, bounds: CGRect(x:
0, y: 0, width: 1, height: 1), format: .RGBA8, colorSpace: CGColorSpaceCreateDeviceRGB())
        } else {

```



```

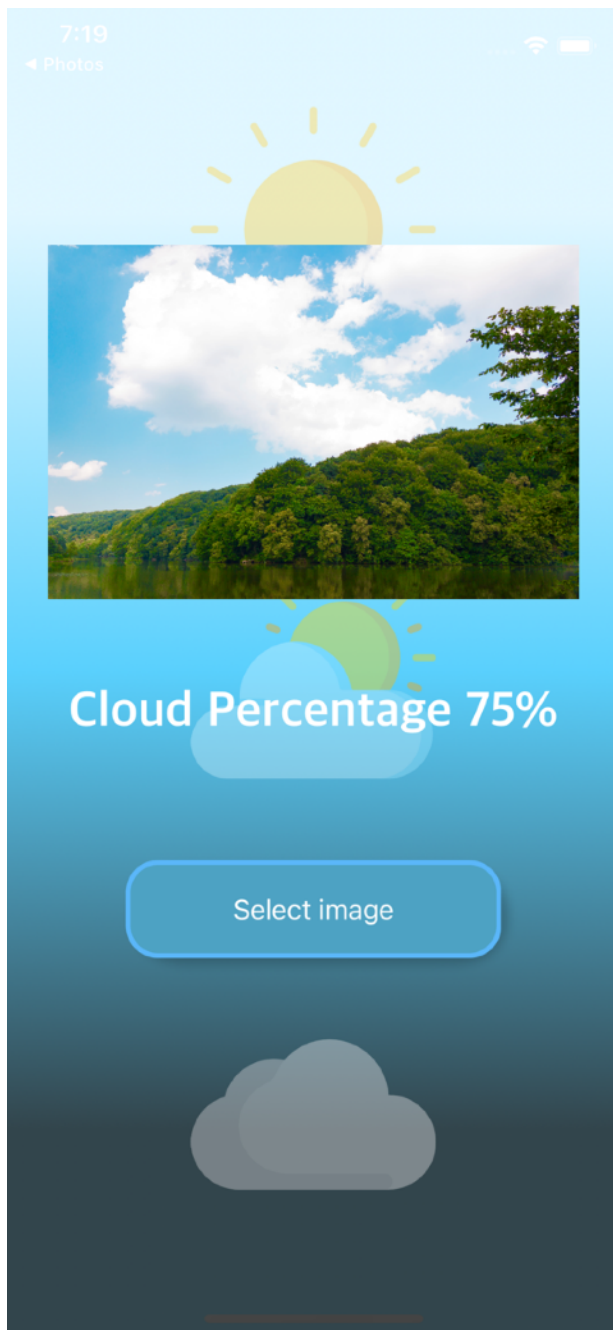
        let context = CGContext(data: &bitmap, width: 1, height: 1, bitsPerComponent: 8,
bytesPerRow: 4, space: CGColorSpaceCreateDeviceRGB(), bitmapInfo:
CGImageAlphaInfo.noneSkipFirst.rawValue!)
        let inputImage = cgImage ?? CIContext().createCGImage(ciImage!, from:
ciImage!.extent)
        context.draw(inputImage!, in: CGRect(x: 0, y: 0, width: 1, height: 1))
    }
    let alpha: CGFloat = CGFloat(bitmap[3]) / 255.0
    let multiplier: CGFloat = alpha*255.0
    return UIColor(red: CGFloat(bitmap[0])/multiplier, green: CGFloat(bitmap[1])/
multiplier, blue: CGFloat(bitmap[2])/multiplier, alpha: alpha)
}
}
extension UIColor {
    var rgba: (red: CGFloat, green: CGFloat, blue: CGFloat, alpha: CGFloat) {
        var red: CGFloat = 0
        var green: CGFloat = 0
        var blue: CGFloat = 0
        var alpha: CGFloat = 0
        getRed(&red, green: &green, blue: &blue, alpha: &alpha)
        return (red, green, blue, alpha)
    }
}
extension Float {
    func correctionValue(by: Float) -> Float {
        if self > by { return 1+(self-by)/by }
        else if self < by { return 1-(by-self)/by }
        else { return 1 }
    }
    func roundedValue(base: Int) -> String {
        let roundedInt = Int(roundf(self))%base < (base/2)+1 ? Int(roundf(self))/base*base :
(Int(roundf(self))/base+1)*base
        return String(roundedInt)+"%"
    }
}
}

```

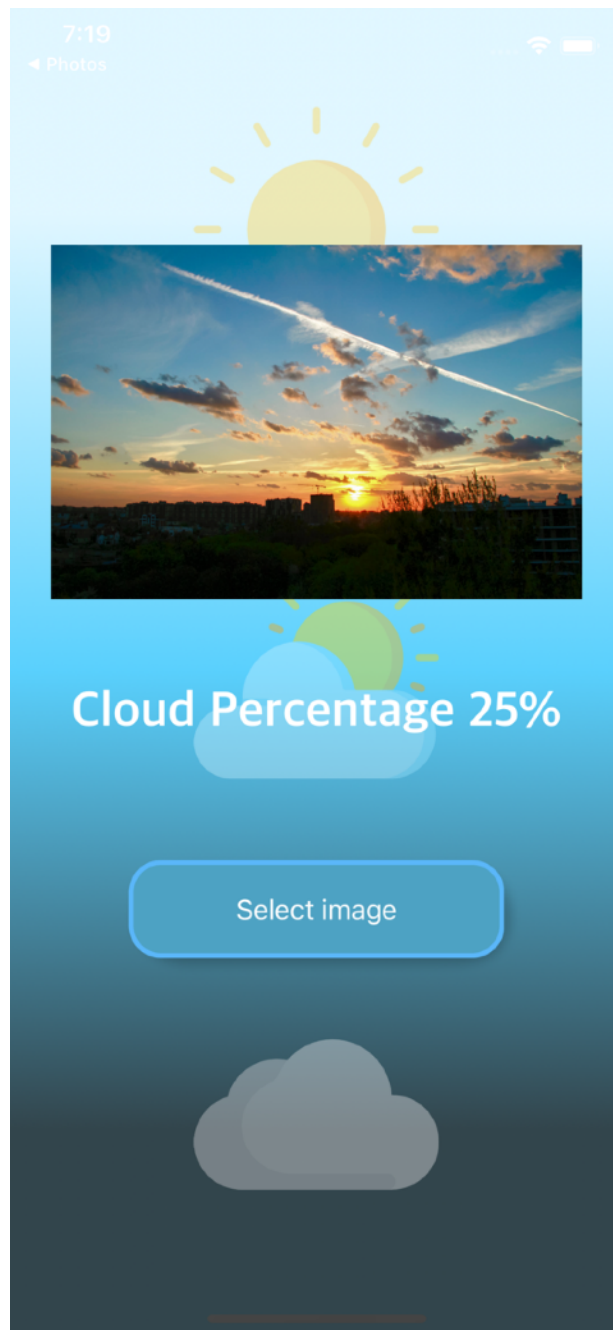
Додаток 2

Результати роботи програми

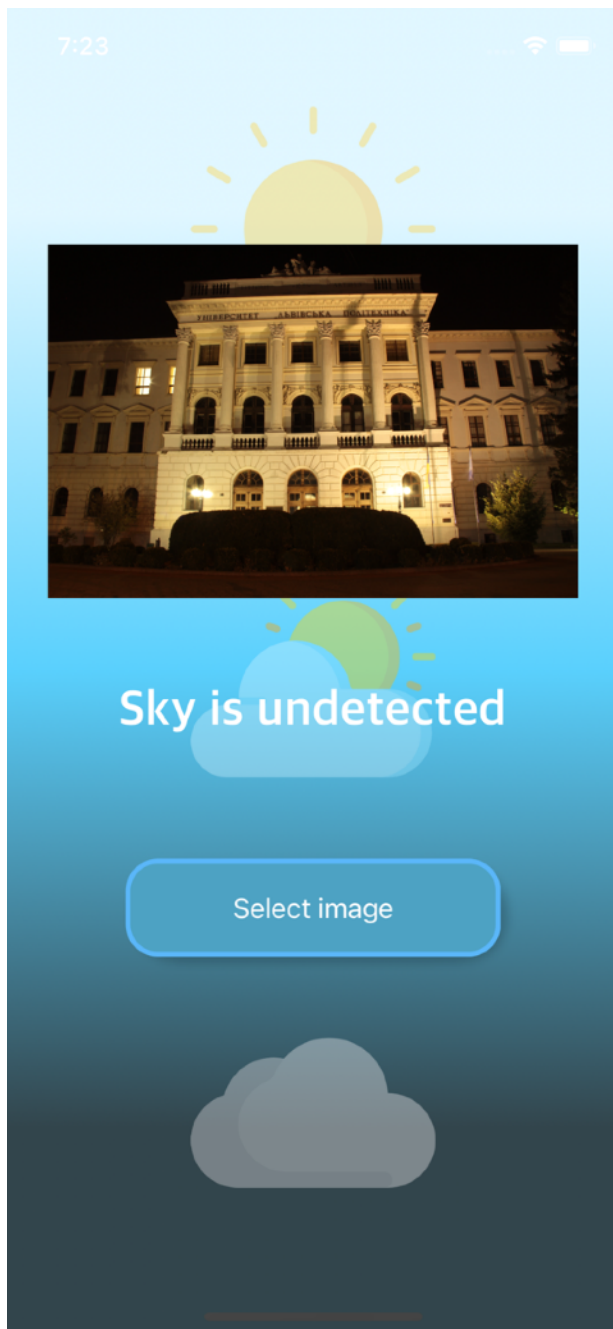
Приклад 1.



Приклад 2.



Приклад 3.



Приклад 4.

