Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования

САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ МОРСКОЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

Факультет цифровых промышленных технологий
Кафедра вычислительной техники и информационных технологий

Отчет

По выполнению практического задания

по теме «**Запись QR-кодов в изображение лица**»

по дисциплине «**Прикладная информатика**».

Специальность: программное обеспечение вычислительной техники и информационных технологий

Выполнил:

Студент группы №20390 *Трапер Максим*

Дата выполнения отчета: 11.05.23

Дата сдачи отчета: 24.05.23

Подпись: _____

Проверил:

Профессор *Щеголева Н.Л.*

Подпись: _____

Санкт-Петербург

2023

Оглавление

Цель работы

Реализация алгоритма генерации BIO-QR кода, встраиваемого в изображение лица, и содержащего в себе биометрическую информацию.

Условие задания

№14 ЗАПИСЬ QR-КОДОВ В ИЗОБРАЖЕНИЕ ЛИЦА

1. **Реализовать приложение для автоматического формирования QR-кодов по изображению лица.**
2. Реализовать разные варианты компоновки информации в рамках «BIO QR-code»
3. Сделать выводы по проделанной работе.

Теоретический материал

QR код (QR - Quick Response) — это двухмерный штрихкод, предоставляющий информацию для быстрого ее распознавания с помощью камеры на мобильном телефоне.

BIO-QR код – QR код, содержащий в себе биометрическую информацию о человеке.

Фенотип – совокупность характеристик, присущих индивиду.

Антропометрические точки - это четко выраженные и легко фиксируемые на теле образования скелета

Ход работы

Ссылка на GitHub:

https://github.com/MaksimTraper/bioqr

Обзор программы:

В стартовом окне программы есть возможность выбрать один из двух режимов работы: ручное формирование BIO-QR или подборка нескольких BIO-QR в заранее заданных компоновках. Реализуется всё на основе выбираемой через проводник фотографии.

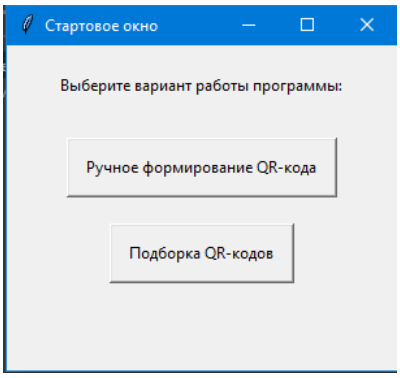


Рис. 1: начальное окно программы

При выборе ручного формирования, предлагается настроить собственную компоновку, основанную на пяти разных элементах: Antro, Info, Pheno, Photo, Scetch. Выбираются параметры через ComboBox`ы.

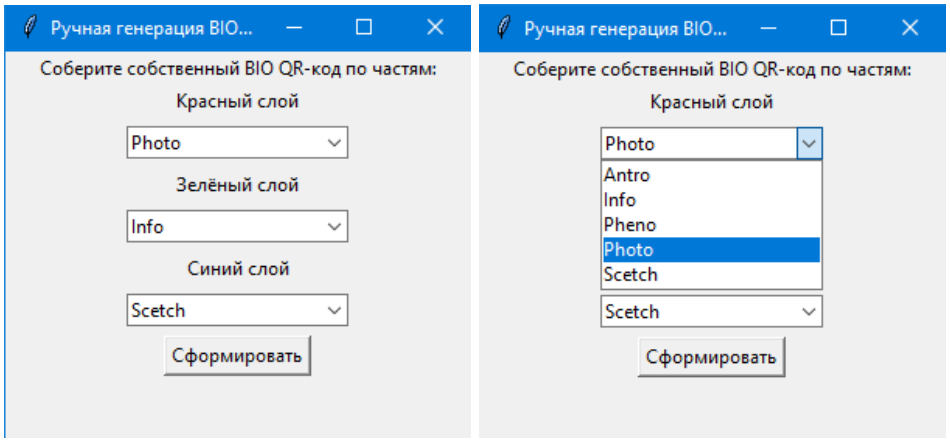


Рис. 2, 3: окно режима ручной генерации BIO-QR. Наполнение ComboBox`а

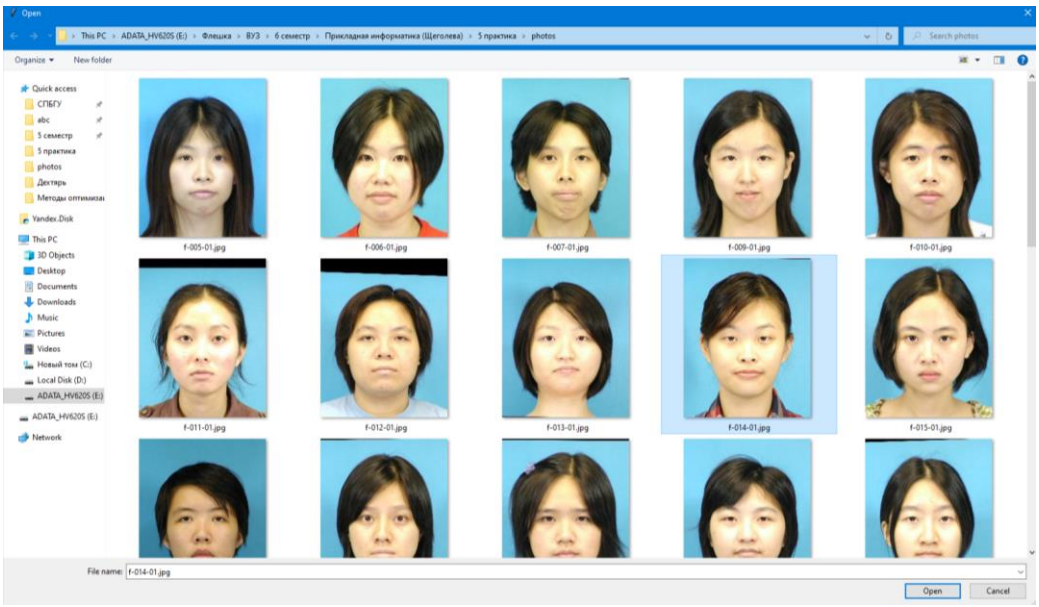


Рис. 4: выбор фотографии для генерации QR-кода

Результат демонстрируется в отдельном окне в виде скомпонованного по выбранным параметрам BIO-QR. Представлен результат на рис. 5.

Рис. 5: результат ручного режима генерации

При выборе иного режима работы, в новом окне появляется исходные и обрезанные фото и скетч, а также 6 разных вариантов компоновки BIO-QR кодов. Все компоновки заведомо заданы в коде программы. Пример представлен на рис. 6.

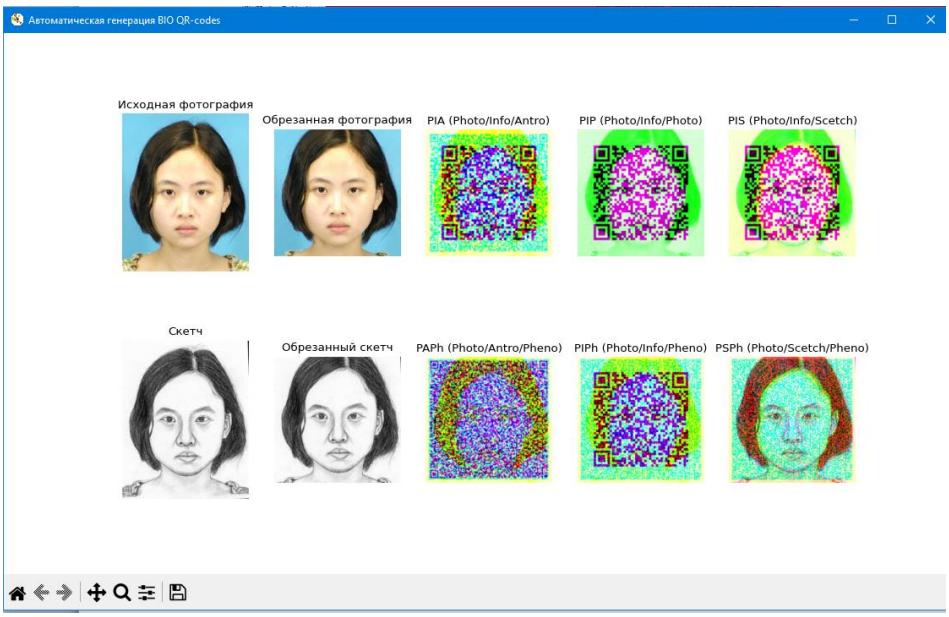


Рис. 6: результат автоматического режима генерации QR-кодов

Код программы:

Программа начинает выполнение с файла main.py. В нём указана только функция генерации стартового окна.

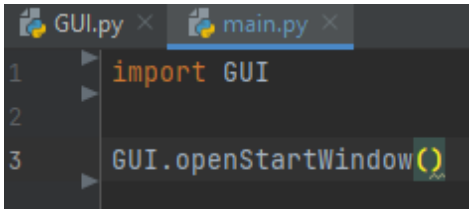
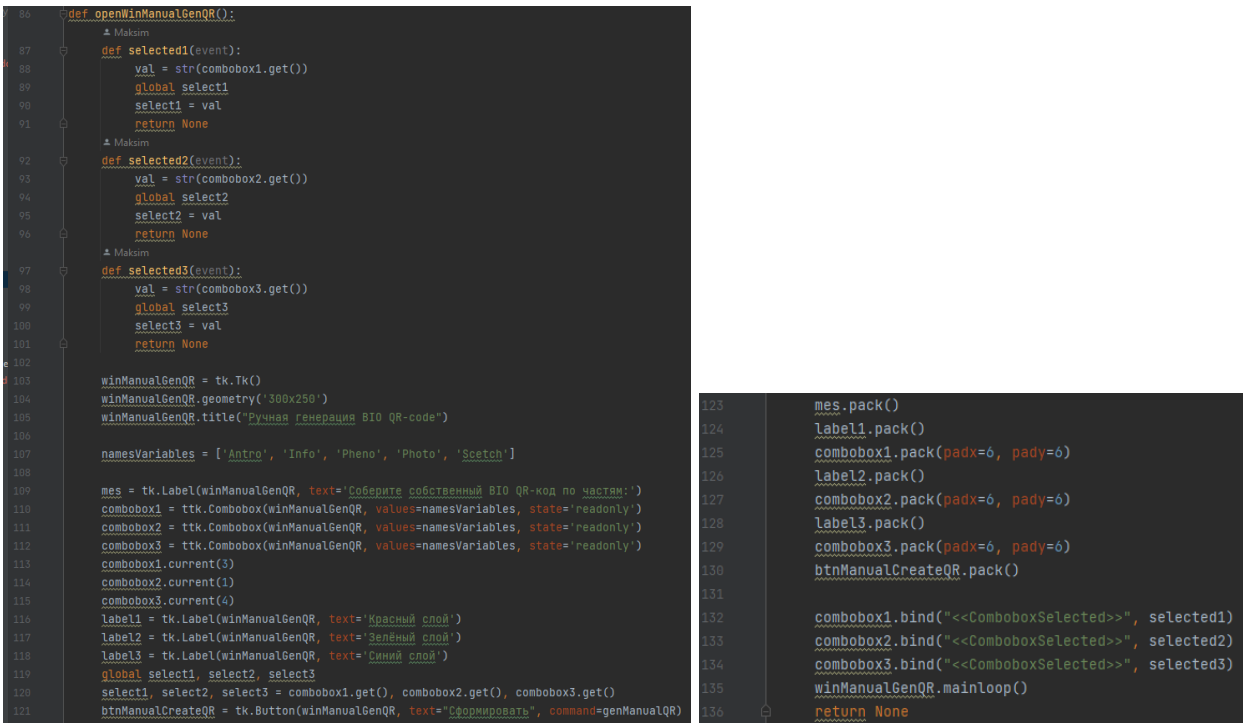


Рис. 7: файл main.py

```
def openStartWindow():
    startwindow = tk.Tk()
    startwindow.geometry('300x250')
    startwindow.title("Стартовое окно")

    mes = tk.Label(startwindow, text = 'Выберите вариант работы программы:')
    btnManualCreateQR = tk.Button(startwindow, text="Ручное формирование QR-кода", command=openWinManualGenQR)
    btnAutoCreateQRs = tk.Button(startwindow, text="Подборка QR-кодов", command=openWinAutoGenQR)

    mes.pack(ipadx=20, ipady=20)
    btnManualCreateQR.pack(ipadx=10, ipady=10, padx=10, pady=10)
    btnAutoCreateQRs.pack(ipadx=10, ipady=10, padx=10, pady=10)

    startwindow.mainloop()
    return None
```

Рис. 8: генерация стартового окна

Режим ручной генерации предполагает создание нового окна и манипуляции с выбранной через проводник фотографией.



```
def openWinManualGenQR():
    Maksim
    def selected1(event):
        val = str(combobox1.get())
        global select1
        select1 = val
        return None
    Maksim
    def selected2(event):
        val = str(combobox2.get())
        global select2
        select2 = val
        return None
    Maksim
    def selected3(event):
        val = str(combobox3.get())
        global select3
        select3 = val
        return None

    winManualGenQR = tk.Tk()
    winManualGenQR.geometry('300x250')
    winManualGenQR.title("Ручная генерация BIO QR-code")

    namesVariables = ['Antro', 'Info', 'Pheno', 'Photo', 'Scetch']

    mes = tk.Label(winManualGenQR, text='Соберите собственный BIO QR-код по частям:')
    combobox1 = ttk.Combobox(winManualGenQR, values=namesVariables, state='readonly')
    combobox2 = ttk.Combobox(winManualGenQR, values=namesVariables, state='readonly')
    combobox3 = ttk.Combobox(winManualGenQR, values=namesVariables, state='readonly')
    combobox1.current(3)
    combobox2.current(1)
    combobox3.current(4)
    label1 = tk.Label(winManualGenQR, text='Красный слой')
    label2 = tk.Label(winManualGenQR, text='Зелёный слой')
    label3 = tk.Label(winManualGenQR, text='Синий слой')
    global select1, select2, select3
    select1, select2, select3 = combobox1.get(), combobox2.get(), combobox3.get()
    btnManualCreateQR = tk.Button(winManualGenQR, text="Сформировать", command=genManualQR)
```

```
    mes.pack()
    label1.pack()
    combobox1.pack(padx=6, pady=6)
    label2.pack()
    combobox2.pack(padx=6, pady=6)
    label3.pack()
    combobox3.pack(padx=6, pady=6)
    btnManualCreateQR.pack()

    combobox1.bind("<<ComboboxSelected>>", selected1)
    combobox2.bind("<<ComboboxSelected>>", selected2)
    combobox3.bind("<<ComboboxSelected>>", selected3)
    winManualGenQR.mainloop()
    return None
```

Рис: 9, 10: генерация окна ручной генерации

Выбранная фотография и её скетч (скетч, в соответствии с фотографией, выбирается автоматически) загружается, как объекты двух библиотек (OpenCV и Pillow).

Далее оба изображения обрезаются, сохраняется на ПК и вновь загружаются, как объект OpenCV. Далее происходит контвертация в полутоновый.

Затем посредством библиотеки dlib фотография преобразуется в объект библиотеки для последующего поиска антропометрических точек.

Далее заполняются три массива Info, Antro, Pheno доступной информацией.

Создаётся объект generator, с полями изображений и массивов. Ищутся антропометрические точки и заполняются оставшейся информацией массивы Antro и Pheno. Генерируются исходные QR-коды, сохраняются и вновь загружаются в систему. В качестве последнего этапа на основе выбранных в окне параметров генерируется BIO-QR код.

```python
def genManualQR():
    photoPath, scetchPath = PhotoManipulating.loadPhotoPath()
    photo, photoPIL = PhotoManipulating.load_photo_2variants(photoPath)
    scetch, scetchPIL = PhotoManipulating.load_photo_2variants(photoPath)
    PhotoManipulating.crop_image(photo, photoPIL, 'photo')
    PhotoManipulating.crop_image(scetch, scetchPIL, 'scetch')
    photo = PhotoManipulating.load_photo('cropped_photo.jpg')
    #photo = cv2.resize(photo, (177, 177))
    gr_photo = PhotoManipulating.cvt_to_gray(photo)

    scetch = PhotoManipulating.load_photo('cropped_scetch.jpg')
    #scetch = cv2.resize(scetch, (177, 177))

    detector = dlib.get_frontal_face_detector()
    faces = detector(gr_photo)
    face = faces[0]

    antro = np.array(['4-May-2023', '/Facial Anthropometric Point', 'Base CUFS: Female f-008-01 /#', [250, 200]],
                     dtype=object)
    info = np.array(['Base CUFS', 'Female f-008-01', 'http://mmlab.ie.cuhk.edu.hk/archive/facesketch.html'])
    pheno = np.array(['4-May-2023', '/Facial Anthropometric Point', 'Photo: 01 /#'])

    generator = GeneratorQR.GeneratorQRCodes(photo, scetch, antro, info, pheno)

    generator.getAntroPhenoMas(photo, face)
    generator.genQRCodes(photo, face, antro, info, pheno)
    generator.loadQRCodes()
    BioQRCode = generator.genBIOQRCodes(select1, select2, select3)

    name = select1[0]+select2[0]+select3[0]+': '+select1+'/'+select2+'/'+select3
    plt.subplots(1, 1, figsize=(7, 7), num='Сгенерированный BIO QR-code')
    plt.imshow(BioQRCode)
    plt.title(name)
    plt.axis('off')

    plt.show()
```

Рис 11: режим ручной генерации BIO-QR

На рис. 12 представлены методы для загрузки фото, выбранного пользователем, в программу.

```python
def loadPhotoPath():
    global photoPath
    photoPath = fd.askopenfilename()
    root_proj = os.getcwd()
    rel_path = os.path.relpath(photoPath, root_proj+'/photos')
    global scetchPath
    scetchPath = up(up(photoPath)) + '/sketches/' + rel_path[0].upper() + '2' + rel_path[1:8:1] + '-sz1.jpg'
    return photoPath, scetchPath

Maksim
def load_photo_2variants(name):
    root_proj = os.getcwd()
    rel_path = os.path.relpath(name, root_proj)
    photo = cv2.imread(rel_path)
    photo = cv2.cvtColor(photo, cv2.COLOR_RGB2BGR)
    photoPIL = Image.open(pathlib.Path(name))
    return photo, photoPIL

Maksim
def load_photo(name):
    photo = cv2.imread(name)
    photo = cv2.cvtColor(photo, cv2.COLOR_RGB2BGR)
    return photo

Maksim
def cvt_to_gray(photo):
    gr_photo = cv2.cvtColor(photo, cv2.COLOR_BGR2GRAY)
    return gr_photo
```

Рис.12: методы загрузки изображений

На рис. 13 показан метод обрезания изображения до разрешения 200 на 200. Выполняется это путём вычисления координат лица на изображении, посредством библиотеки dlib, затем симметричным расширением выделенной области до необходимых значений.

На рис. 14 показан метод разложения изображения на RGB-слои. Используется метод array_split библиотеки Numpy.

```python
def crop_image(photo, photoPIL, name):
    # Загрузка компонентов, обнаруживающих АПТ и из запись
    detector = dlib.get_frontal_face_detector()
    gr_photo = cvt_to_gray(photo)
    faces = detector(gr_photo)
    face = faces[0]
    width = face.right() - face.left()
    height = face.bottom() - face.top()
    differ = 200 - width
    right = face.right() + differ / 2
    top = face.top() - differ / 6 * 3
    left = face.left() - differ / 2
    bottom = face.bottom() + differ / 6
    while ((right - left) != 200 or (bottom - top) != 200):
        if ((right - left) != 200):
            if (right - left) > 200:
                right -= 1
            else:
                right += 1
        if ((bottom - top) != 200):
            if (bottom - top) > 200:
                top += 1
            else:
                top -= 1
    cropped_image = photoPIL.crop((left, top, right, bottom))
    if(name == 'photo'):
        cropped_image.save('cropped_photo.jpg')
    else:
        cropped_image.save('cropped_scetch.jpg')
    return cropped_image
```

Рис. 13: метод обрезки фотографии до размеров $200x200$

```python
def dividePhotoOnRGB(photo):
    b, g, r = np.array_split(photo, 3, axis=2)
    return r, g, b
```

Рис. 14: метод разложения изображения на RGB-компоненты

На рис. 15 показан метод получения информации о фенотипе (координаты антропометрических точек и значения яркости в каждом слои в этих точках). Для поиска точек используется библиотека dlib и обученная модель.

```python
def getAntroPhenoMas(self, photo, face):
    gr_photo = PhotoManipulating.cvt_to_gray(photo)
    predictor = dlib.shape_predictor("shape_predictor_68_face_landmarks.dat")
    landmarks = predictor(gr_photo, face)
    x = np.array([])
    y = np.array([])
    b_br = np.array([])
    g_br = np.array([])
    r_br = np.array([])
    for n in range(0, 68):
        x = np.append(x, landmarks.part(n).x)
        y = np.append(y, landmarks.part(n).y)
        b_br = np.append(b_br, self.b[int(y[n]), int(x[n])])
        g_br = np.append(g_br, self.g[int(y[n]), int(x[n])])
        r_br = np.append(r_br, self.r[int(y[n]), int(x[n])])
    xy = np.concatenate((x, y))
    bright_coord = np.concatenate((r_br, g_br, b_br))
    return xy, bright_coord
```

Рис.15: получение данных о фенотипе

На рис. 16 показан метод, генерирующий QR_{INFO}, QR_{ANTRO}, QR_{PHENO} в качестве отдельных QR-кодов. Затем они сохраняются в виде изображений в проекте. Для этого использована библиотека “qrcode”.

```python
def genQRCodes(self, photo, face, antroIn, infoIn, phenoIn):
    INFO_QR = qrcode.QRCode(box_size=3, border=7)
    ANTRO_QR = qrcode.QRCode(box_size=1)
    PHENO_QR = qrcode.QRCode(box_size=1)

    ANTRO_QR.add_data(self.antro)
    ANTRO_QR.make(fit=True)
    INFO_QR.add_data(self.info)
    INFO_QR.make(fit=True)
    PHENO_QR.add_data(self.pheno)
    PHENO_QR.make(fit=True)

    #Алгоритм увеличения разрешения каждого QR-кода >200 на 200
    #QRs = [INFO_QR, ANTRO_QR, PHENO_QR]
    #for i in range(0, 3, 1):
    #    resolution = (17 + 4*QRs[i].version + QRs[i].border*2)*QRs[i].box_size
    #    add = 0
    #    while(resolution<200):
    #        add += 1
    #        resolution = (17 + 4*QRs[i].version + QRs[i].border*2)*(QRs[i].box_size+add)
    #    QRs[i].box_size += add-1
    #    resolution = (17 + 4 * QRs[i].version + QRs[i].border*2) * QRs[i].box_size
    #    add = 0
    #    while(resolution<200):
    #        add += 1
    #        resolution = (17 + 4 * QRs[i].version + (QRs[i].border+add)*2) * QRs[i].box_size
    #    QRs[i].border += add

    Img_INFO_QR = INFO_QR.make_image()
    Img_INFO_QR.save('INFO_QR.jpg')
    Img_ANTRO_QR = ANTRO_QR.make_image()
    Img_ANTRO_QR.save('ANTRO_QR.jpg')
    Img_PHENO_QR = PHENO_QR.make_image()
    Img_PHENO_QR.save('PHENO_QR.jpg')
    return None
```

Рис. 16: генерация QR-кодов

На рис. 17 показан основной метод программы, компонующий отдельные элементы в BIO-QR код.

```python
def genBIOQRCodes(self, nameOnRed, nameOnGreen, nameOnBlue):
    namesVariables=['Antro', 'Info', 'Pheno', 'Photo', 'Scetch']
    namesParameters = [nameOnRed, nameOnGreen, nameOnBlue]
    #Заполняем массив фиктивными переменными. В последствии, часть из них будут заменены
    parameters = [self.grPhoto, self.grPhoto, self.grPhoto]
    for y in range(0,3,1):
        match namesParameters[y]:
            case 'Antro':
                parameters[y] = self.gr_Img_ANTRO_QR
            case 'Info':
                parameters[y] = self.gr_Img_INFO_QR
            case 'Pheno':
                parameters[y] = self.gr_Img_PHENO_QR
            case 'Photo':
                parameters[y] = self.grPhoto
            case 'Scetch':
                parameters[y] = self.grScetch
    r = cv2.addWeighted(self.r, 0, parameters[0], 1, 0)
    g = cv2.addWeighted(self.g, 0, parameters[1], 1, 0)
    b = cv2.addWeighted(self.b, 0, parameters[2], 1, 0)
    BioQRCode = cv2.merge((b, g, r))
    return BioQRCode
```

Рис. 17: генерация BIO-QR кода

Анализ результатов работы программы:

Попробуем отсканировать сгенерированные BIO-QR коды. Для этого воспользуемся мобильной программой Google Объектив.

В обоих случаях наличия в BIO-QR одного QR или двух QR, с усилиями, смог считаться контрастный QR_{INFO}. Результат представлен на рис. 18 и 19.

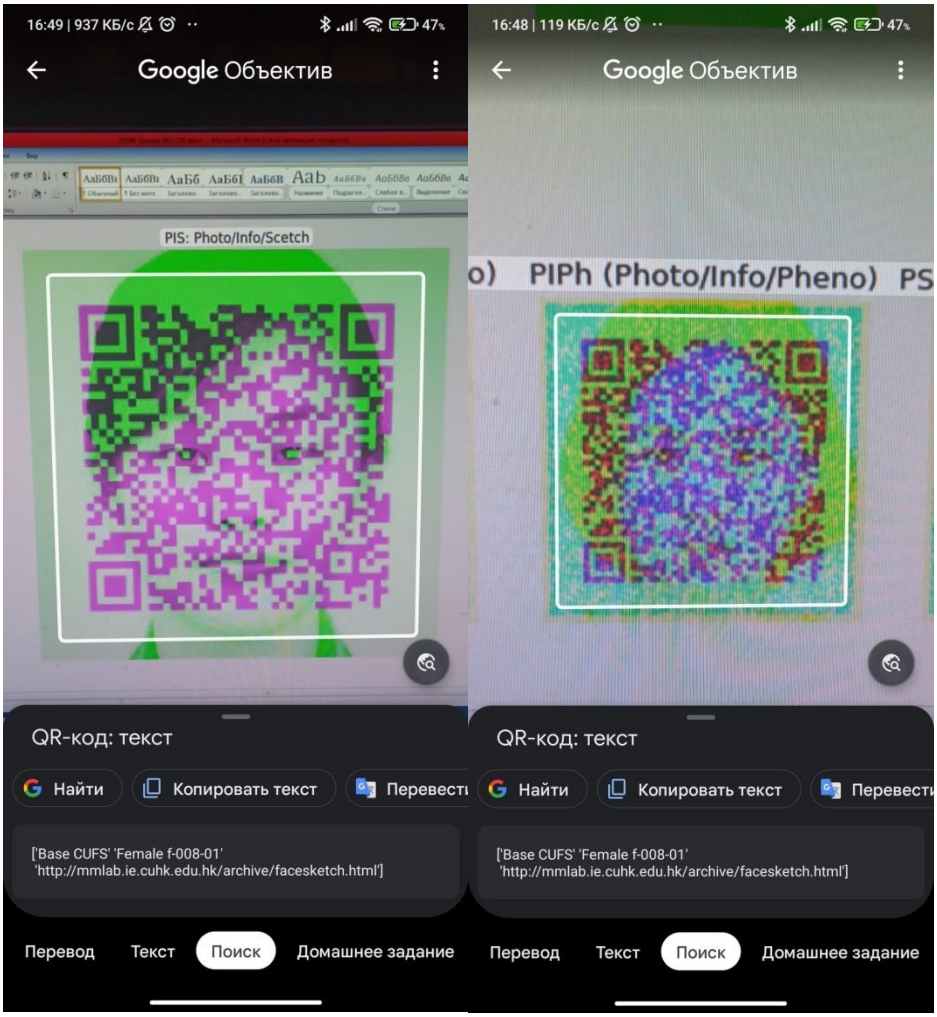


Рис. 18, 19

В случае, когда в BIO-QR оказались и QR_{ANTRO}, и QR_{PHENO} считать ни один из QR кодов не удалось. Показано это на рис. 20. Произошло это, вероятно, из-за смешения картинки и невозможности поверхностного сканирования. Чтобы полутить информацию, записанную в BIO-QR, необходимо его раскладывать на слои и выводить слои как отдельные изображения, чтобы было возможно считать заложенную информацию.

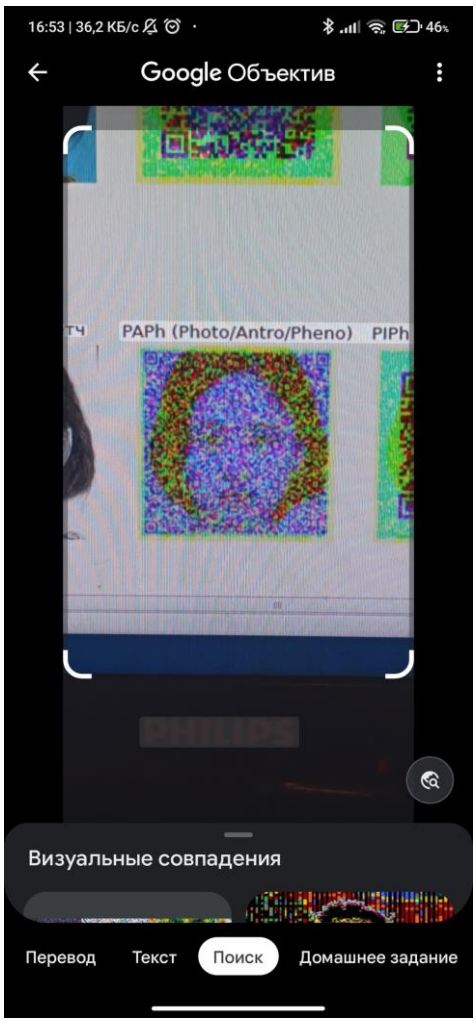


Рис. 20

Вывод:

В рамках данной работы был реализован алгоритм генерации BIO-QR кода, содержащего в себе биометрическую информацию. Признана его эффективность, как единой структуры для передачи информации о человеке, но, в большинстве случаев, необходимы дополнительные манипуляции для получения всей зашифрованной информации в BIO-QR коде, что является минусом.