Министерство науки и высшего образования Российской Федерации
**Муромский институт (филиал)**
федерального государственного бюджетного образовательного учреждения высшего образования
**«Владимирский государственный университет
имени Александра Григорьевича и Николая Григорьевича Столетовых»**
**(МИ ВлГУ)**

Факультет ИТР
Кафедра ИС

# *ОТЧЕТ*

По НИР

тема: «Сопоставление растровой и векторной информации»

Руководитель

к. т. н., доц. каф. ИС
(уч. степень, звание)

Еремеев С. В.
(фамилия, инициалы)

(оценка)

(подпись) (дата)

Члены комиссии

Студент ИС-118
(группа)

Панкратов Д.А.
(фамилия, инициалы)

(подпись) (Ф.И.О.)

(подпись) (Ф.И.О.)

(подпись) (дата)

Муром 2022

СОДЕРЖАНИЕ

# 1 Постановка задачи

В данной исследовательской работе была поставлена задача разработать алгоритм сопоставления растровой и векторной карты, определённой области, в независимости от смещения и угла поворота.

Исходные данные алгоритма:

– Идентичные векторная и растровая карты;

– Смещённая и/или повёрнутая растровая карта той же области.

В качестве среды разработки будет использоваться свободная кроссплатформенная геоинформационная система QGIS, используемая для создания, редактирования, визуализации, анализа и публикации геопространственной информации. QGIS работает в Windows и в большинстве платформ Unix (включая Mac OS), поддерживает множество векторных и растровых форматов и баз данных, а также имеет богатый набор встроенных инструментов.

Для реализации алгоритма будет использоваться язык программирования PyQGIS. Данный язык позволяет запускать созданный скрипт в консоли системы QGIS, что ускорит и упростит процесс реализации алгоритма.

2 Разработка алгоритма

Для реализации поставленной задачи было создано несколько файлов кода:

1 Файл предобработки: преобразует растровое изображение в бинарное, оставляя только дома:

1.1 Растровое изображение разделяем на 3 канала;

1.2 Берём 2 канал (G);

1.3 Все пиксели полученного канала яркость которых больше 238 преобразуем в 0, остальные в значение яркости 255;

1.4 Изображение подвергаем медиальному сглаживанию (cv2.mediaBlur(img,5));

1.5 К полученным изображениям в 4 и 2 пунктах применяем операцию побитового и, для получения зданий;

1.6 Находим контура зданий (cv2.findContours(img, cv2.RETR_LIST, cv2.LINE_4));

1.7 Полученные контура рисуем на новом изображении;

1.8 К полученному изображению добавляем рамку в 20 px, для получения зданий на краю фото при создании растрового слоя;

1.9 Полученное изображение сохраняем.

2 Файл создания векторного слоя:

2.1 Полученное после предобработки изображение разделяем на 3 канала;

2.2 Берём 1 канал (R);

2.3 Все пиксели полученного канала яркость которых больше 238 преобразуем в 0, остальные в значение яркости 255;

2.4 Изображение подвергаем медиальному сглаживанию (cv2.mediaBlur(img,3));

2.5 Находим конура зданий (cv2.findContours(img, cv2.RETR_LIST, cv2.LINE_4));

2.6 Цикл по контурам:

2.6.1 Если контур является последним, то выходим из цикла;

2.6.2 Создаём объект (далее рассматриваемый объект);

2.6.3 Находим площадь контура и определяем его точки;

2.6.4 Используем точки контура для создания геометрии;

2.6.5 Применяем созданную геометрию к объекту;

2.6.6 Если контур является первым, то добавляем его геометрию в список геометрий, которые необходимо отобразить и переходим к следующей итерации цикла;

2.6.7 Цикл по списку объектов, которые необходимо отобразить:

2.6.7.1 Сравниваем объект, который необходимо отобразить с рассматриваемым объектом;

2.6.7.2 Если площадь объекта, который необходимо отобразить меньше или равна площади рассматриваемого объекта:

2.6.7.2.1 Цикл по точкам объекта, который необходимо отобразить, в котором проверяем входят ли все точки этого объекта в рассматриваемый объект;

2.6.7.2.2 Если да, то заносим в список для удаления объект, который необходимо отобразить и выходим из цикла;

2.6.7.2.3 Иначе переходим к следующей итерации цикла;

2.6.7.3 Иначе:

2.6.7.3.1 Цикл по точкам рассматриваемого объекта, в котором проверяем входят ли все точки этого объекта, в объект, который необходимо отобразить;

2.6.7.3.2 Если да, то выходим из цикла;

2.6.7.3.3 Иначе переходим к следующей итерации цикла;

2.6.8 Если рассматриваемый объект не находится внутри другого объекта, то добавляем его в список объектов, которые необходимо отобразить;

2.7 Удаляем из списка объектов, которые необходимо отобразить, объекты, полученные в пункте 2.6.7.2.2;

2.8 Отображаем полученные объекты;

3 Файл сопоставления растровой и векторной информации:

3.1 Загружаем информацию о векторном слое;

3.2 Загружаем идентичную растровой векторную карту и векторную карту, с которой необходимо произвести сопоставление;

3.3 Находим ключевые точки загруженных карт (sift.detectAndCompute);

3.4 Сопоставляем полученные точки;

3.5 Берём 2 пары наиболее точно сопоставленных ключевых точек по значению идентичности;

3.6 Рассчитываем расстояние по x и по y от одной ключевой точки, до другой для векторной карты (рис. 1);

3.7 Рассчитываем расстояние по x и по y от одной ключевой точки, до другой для растровой карты (рис. 1);

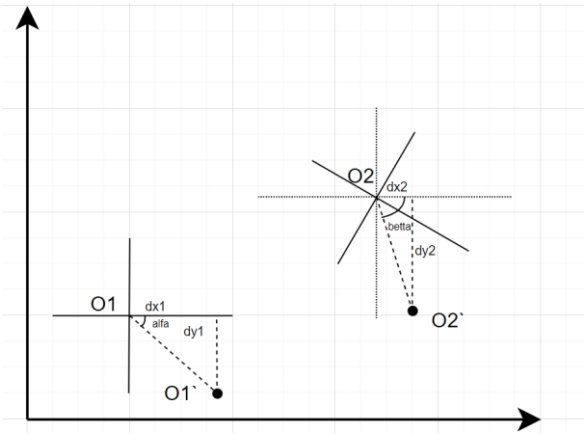


Рисунок 1 – Расчёт угла поворота

3.8 Находим угол alfa и угол betta:

$$alfa = \tan(\frac{dy_1}{dx_1}) \quad (1)$$

где $dx_1$ и $dy_1$ - расстояние по x и по y от одной ключевой точки, до другой для векторной карты.

$$betta = \tan(\frac{dy_2}{dx_2}) \quad (2)$$

где $dx_2$ и $dy_2$ - расстояние по x и по y от одной ключевой точки, до другой для растровой карты.

3.9 Получаем угол поворота растровой карты, вычитанием угла alfa из угла betta;

3.10 Цикл по объектам векторного слоя:

3.10.1 Цикл по точкам объекта:

3.10.1.1 Рассчитываем расстояние от ключевой точки векторного изображения до точки объекта;

3.10.1.2 Получаем расстояние от ключевой точки растрового изображения до точки объекта, сложением координат ключевой точки и значения, полученного в пункте 3.10.1.1;

3.10.1.3 Рассчитываем координаты новой точки объекта с учётом угла поворота:

$$x` = x_0 + (x - x_0)\cos\alpha - (y - y_0)\sin\alpha \quad (3)$$

$$y` = y_0 + (x - x_0)\sin\alpha + (y - y_0)\cos\alpha \quad (4)$$

где $x_0$ и $y_0$ - координаты ключевой точки растрового изображения

x и y – координаты полученные в пункте 3.10.1.3;

α – угол поворота растровой карты.

3.10.2 Отображение объекта с новыми кордонами.

3 Исследовательская часть

В ходе исследовательской работы данный алгоритм был протестирован на реальных примерах:

Рисунок 2 – Исходная векторная карта

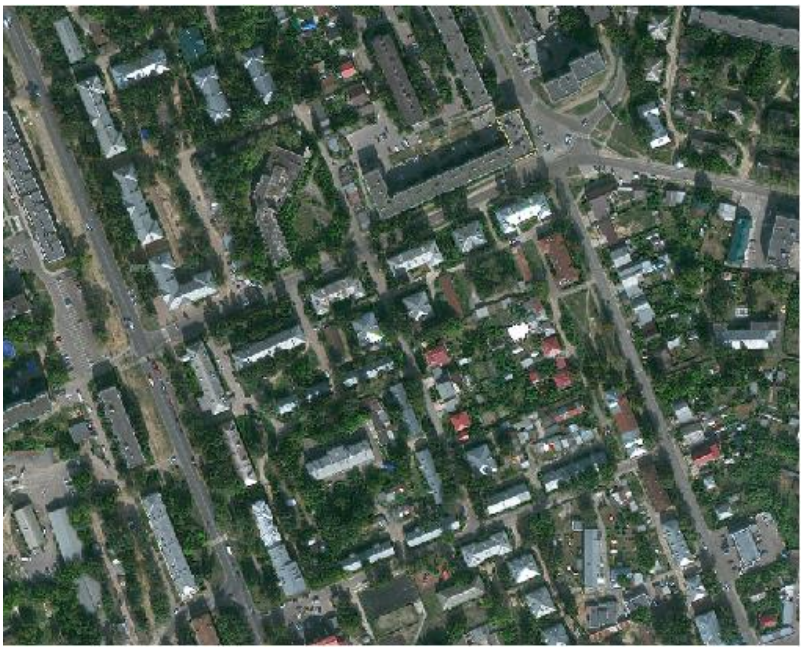

Рисунок 3 – Исходная растровая карта, идентичная векторной

1 Сопоставление векторной и смещённой растровой карты:

Рисунок 4 – Смещённая растровая карта

Рисунок 5 – Сопоставление ключевых точек

Координаты ключевых точек на векторном изображении равны:

- 2.590 по x и 168.678 по y.

- 3.0967 по x и 220.317 по y.

Координаты ключевых точек на растровом изображении равны:

- 2.590 по x и 168.678 по y.

- 3.0967 по x и 220.317 по y.

Рисунок 6 – Сопоставление растровой и векторной карты

2 Сопоставление векторной и повёрнутой растровой карты:

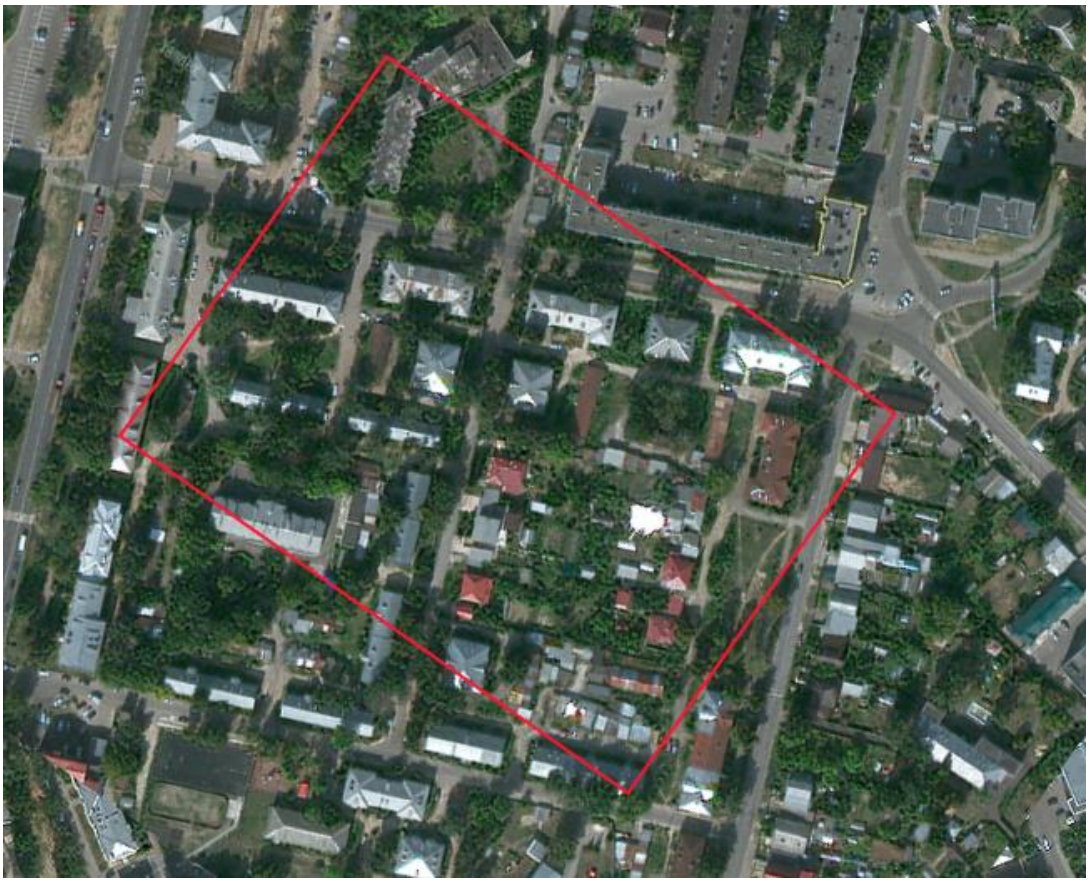

Рисунок 7 – Повёрнутая растровая карта на 35 градусов

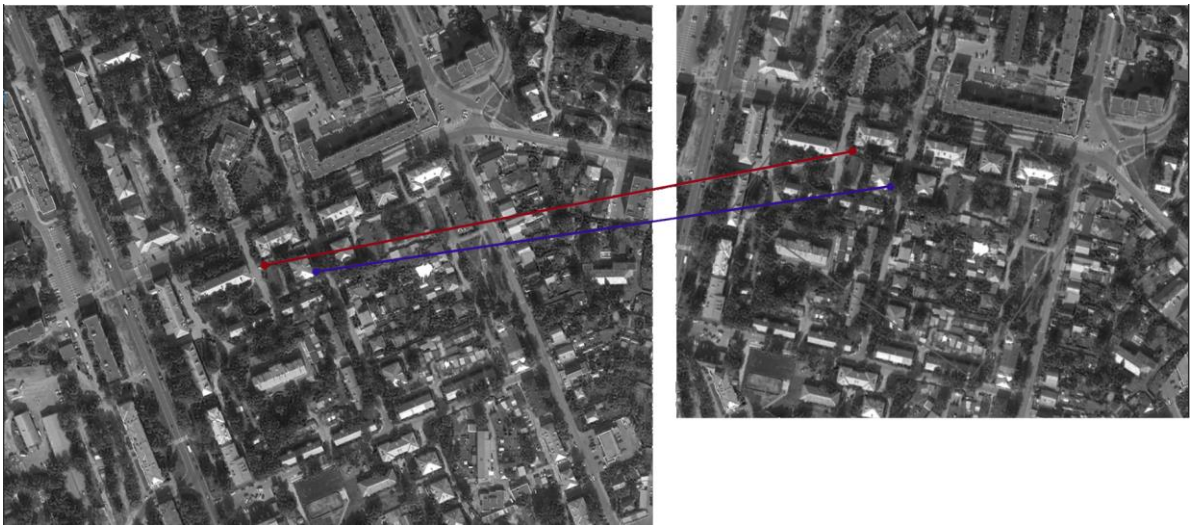

Рисунок 8 – Сопоставление ключевых точек

Координаты ключевых точек на векторном изображении равны:

- 329.125 по x и 328.061 по y.
- 393.933 по x и 336.231 по y.

Координаты ключевых точек на растровом изображении равны:

- 223.034 по x и 185.133 по y.
- 271.434 по x и 229.111 по y.

Рисунок 9 – Сопоставление растровой и векторной карты

3 Сопоставление векторной и смещённой и повёрнутой растровой карты:

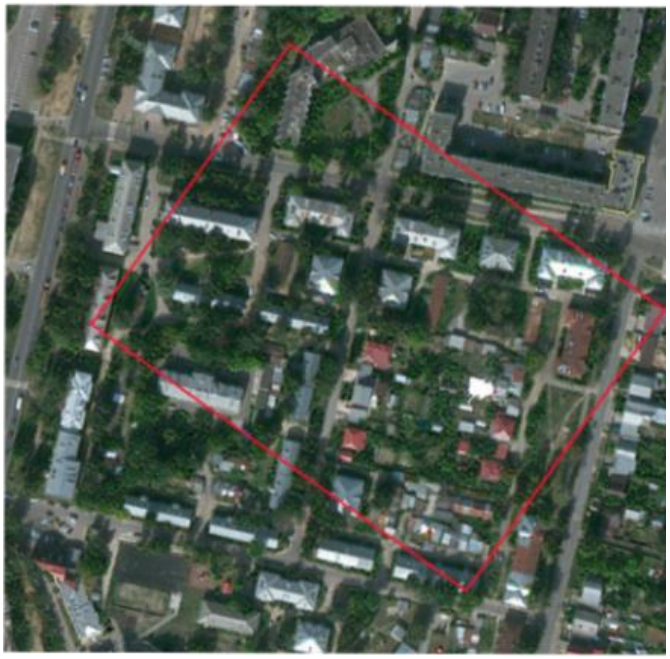

Рисунок 10 – Смещённая и повёрнутая на 35 градусов растровая карта

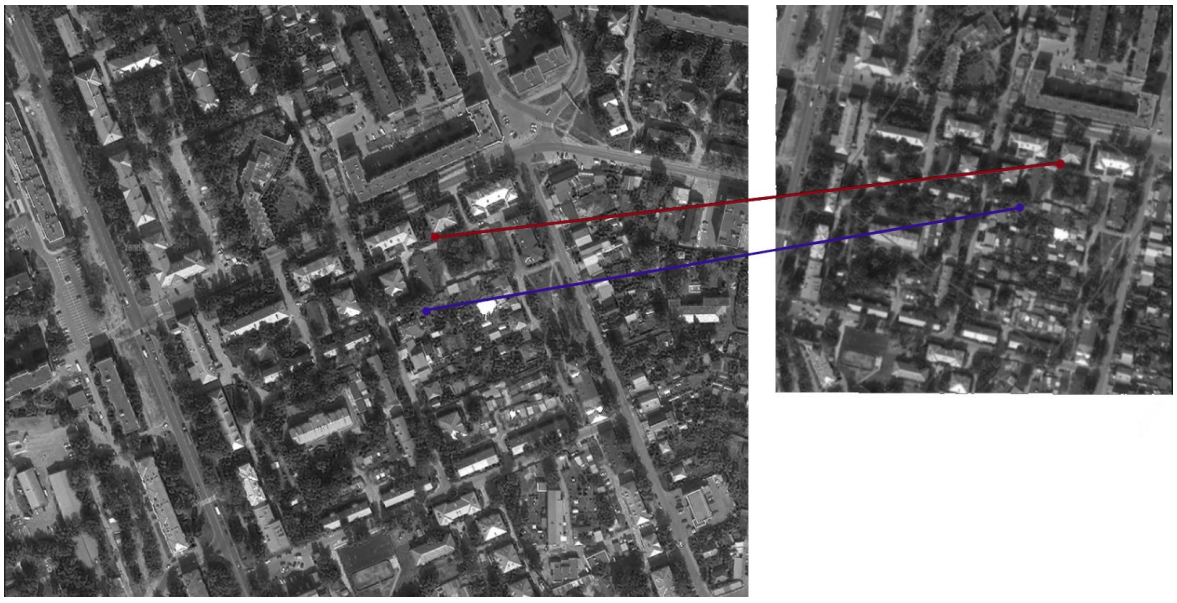

Рисунок 11 – Сопоставление ключевых точек

Координаты ключевых точек на векторном изображении равны:

- 475.601 по x и 258.221 по y.

- 465.524 по x и 338.502 по y.

Координаты ключевых точек на растровом изображении равны:

- 313.761 по x и 177.098 по y.

- 269.620 по x и 226.044 по y.

Рисунок 12 – Сопоставление растровой и векторной карты

В ходе тестирования алгоритма, ошибок выявлено не было.

# ЗАКЛЮЧЕНИЕ

В ходе выполнения исследовательской работы был реализован алгоритм Сопоставления растровой и векторной информации, удовлетворяющий всем условиям, поставленным в техническом задание.

Проведенное исследование показывает, что данный алгоритм работает правильно в независимости от угла поворота и обрезанной части векторной карты.

Время выполнения алгоритма составляет менее 1 секунды и зависит от:

- Количества домов на карте;
- Характеристик используемого аппаратного обеспечения.

СПИСОК ЛИТЕРАТУРЫ

1) Основы геоинформатики: В 2 кн. Кн. 1: Учеб. пособие для 0-75 студ. вузов / Е.Г.Капралов, А.В.Кошкарев, В.С.Тикунов и др.; Под ред. В.С.Тикунова. — М.: Издательский центр «Академия», 2004. — 352 е., [16] с.

2) Документация QGIS// URL: https://qgis.org/ru/docs/index.html (дата обращения: 10.02.2022)

3) Документация PyQGIS// URL: https://qgis.org/pyqgis/ (дата обращения: 30.01.2022)

# ПРИЛОЖЕНИЕ

## Файл предобработки изображения:

```
import staticmaps
import cv2
import numpy as np
import os
import random as rng


img = cv2.imread('E:/5_set.png')


sourceColor = np.array(img)[:-15,:]
layers = cv2.split(sourceColor)
imgnp = layers[1]

_,imgnp = cv2.threshold(imgnp, 238, 255, cv2.THRESH_BINARY_INV)
imgnp_med = cv2.medianBlur(imgnp, 5)
imgnp = cv2.bitwise_and(imgnp_med, imgnp)

contours, _ = cv2.findContours(imgnp, cv2.RETR_LIST, cv2.LINE_4)

drawing = np.zeros            9_like(imgnp)
for cnt in contours:
     M = cv2.moments(cnt)
     if M['m00'] == 0:
         continue

     if cv2.contourArea(cnt) < cv2.arcLength(cnt,True):
          continue

     cv2.drawContours(drawing, [cnt], 0, (255), cv2.FILLED)

drawing = cv2.bitwise_and(imgnp, drawing)
img_new = cv2.copyMakeBorder(drawing, 20, 20, 20, 20, cv2.BORDER_CONSTANT)
cv2.imwrite(os.path.join(dir, str(k) + '_contr.png'), img_new)
```

## Файл создания векторного слоя:

```
import cv2
import numpy as np

img = cv2.imread('E:/Lab/4course2s/NIR/vec/tiles/5_contr.png')
#разделяем изображение на 3 канала
layers = cv2.split(img)
imgnp = layers[1]


#преобразуем все пиксели изображения яркость которых выше 238 в 0, остальные в
значение яркости 255
_,imgnp = cv2.threshold(imgnp, 238, 255, cv2.THRESH_BINARY_INV)


imgnp_med = cv2.medianBlur(imgnp, 3)
#imgnp = cv2.bitwise_and(imgnp_med, imgnp)
```

```
contours, _ = cv2.findContours(imgnp_med, cv2.RETR_LIST, cv2.LINE_4)



listGeom = []
i = 0
j = 0
suri = "MultiPolygon?crs=epsg:20008&index=yes"
vl = QgsVectorLayer(suri, "treangle0", "memory")
removeList = []
for count in contours:
    if (i < (len(contours) - 1)):
        i+=1
        fet = QgsFeature()
        epsilon = 0.0001 * cv2.arcLength(count, True)
        approximations = cv2.approxPolyDP(count, epsilon, True)
        a = []
        b = []
        for point in approximations:
            b.append(QgsPointXY(point[0][0] -20, -point[0][1] + 20))
        b.append(QgsPointXY(approximations[0][0][0]-20,-
approximations[0][0][1]+20))
        a.append(b)
        fet.setGeometry(QgsGeometry.fromPolygonXY(a))
        if ( j == 0):
            j+=1
            listGeom.append(fet.geometry())
        else:
            checkGeom = 0
            for geomF in listGeom:
                if not geomF.equals(fet.geometry()):
                    if (geomF.area() <= fet.geometry().area()):
                        '''
                        if geomF.within(fet.geometry()):
                            checkGeom = 1
                            removeList.append(geomF)
                            continue
                        else:
                            checkGeom = 2
                            continue
                            '''
                        countPoints = 0
                        for point in geomF.asPolygon()[0]:
                            if fet.geometry().contains(point):
                               countPoints += 1
                        if countPoints == len(geomF.asPolygon()[0]):
                            checkGeom = 1
                            removeList.append(geomF)
                            continue
                        else:
                            checkGeom = 2
                            continue
                    else:
                        '''
                        if fet.geometry().within(geomF):
                            checkGeom = 3
                            break
                        else:
                            checkGeom = 2
                            continue
                            '''
                        countPoints = 0
                        for point in fet.geometry().asPolygon()[0]:
```

```
                            if geomF.contains(point):
                                countPoints += 1
                        if countPoints == len(fet.geometry().asPolygon()[0]):
                            checkGeom = 3
                            break
                        else:
                            checkGeom = 2
                            continue
            if checkGeom == 1:
                listGeom.append(fet.geometry())
            elif checkGeom == 2:
                listGeom.append(fet.geometry())
            elif checkGeom == 3:
                continue



for geom in removeList:
    if geom in listGeom:
        listGeom.remove(geom)


suri = "MultiPolygon?crs=epsg:20008&index=yes"
vl = QgsVectorLayer(suri, "treangle1" , "memory")
for geomF in listGeom:
    fet = QgsFeature()
    pr = vl.dataProvider()
    vl.updateExtents()
    fet.setGeometry(geomF)
    pr.addFeatures([fet])
    vl.updateExtents()

    if not vl.isValid():
        print("Layer failed to load!")
    else:
        QgsProject.instance().addMapLayer(vl)
```

Файл сопоставления растровой и векторной информации:

```
import math

project = QgsProject.instance()
layer = project.mapLayersByName("treangle1")[0]
print(layer)
feat = layer.getFeatures()

#нахождение ключевых точек

img1 = cv2.imread('E:/Lab/4course2s/NIR/vec/tiles/5_set.png')
img2 = cv2.imread('E:/Lab/4course2s/NIR/vec/tiles1/5_set3.png')

img1 = cv2.cvtColor(img1, cv2.COLOR_BGR2GRAY)
img2 = cv2.cvtColor(img2, cv2.COLOR_BGR2GRAY)
# create SIFT object
sift = cv2.SIFT_create()
# detect SIFT features in both images
keypoints_1, descriptors_1 = sift.detectAndCompute(img1,None)
keypoints_2, descriptors_2 = sift.detectAndCompute(img2,None)
# create feature matcher
bf = cv2.BFMatcher(cv2.NORM_L1, crossCheck=True)
# match descriptors of both images
matches = bf.match(descriptors_1,descriptors_2)
# sort matches by distance
matches = sorted(matches, key = lambda x:x.distance)

matchesList = []
idMat = -1
for mat in matches:
    if abs(keypoints_1[mat.queryIdx].response -
keypoints_2[mat.trainIdx].response) < 0.00001:
        if(len(matchesList) == 1  and
(abs(keypoints_1[matchesList[0].queryIdx].pt[0] -
keypoints_1[mat.queryIdx].pt[0]) > 10) )  :
            continue
        matchesList.append(mat)
        if len(matchesList) == 2 :
            break

pointSwichesVectX1 = keypoints_1[matchesList[0].queryIdx].pt[0]
pointSwichesVectY1 = -keypoints_1[matchesList[0].queryIdx].pt[1]
pointSwichesVectX2 = keypoints_1[matchesList[1].queryIdx].pt[0]
pointSwichesVectY2 = -keypoints_1[matchesList[1].queryIdx].pt[1]

pointSwichesRastX1 = keypoints_2[matchesList[0].trainIdx].pt[0]
pointSwichesRastY1 = -keypoints_2[matchesList[0].trainIdx].pt[1]
pointSwichesRastX2= keypoints_2[matchesList[1].trainIdx].pt[0]
pointSwichesRastY2 = -keypoints_2[matchesList[1].trainIdx].pt[1]

print(pointSwichesVectX1)
print(pointSwichesVectY1)
print(pointSwichesVectX2)
print(pointSwichesVectY2)
print(pointSwichesRastX1)
print(pointSwichesRastY1)
print(pointSwichesRastX2)
print(pointSwichesRastY2)
#

Dx1 = pointSwichesVectX2 - pointSwichesVectX1
```

```
Dy1 = pointSwichesVectY2 - pointSwichesVectY1

Dx2 = pointSwichesRastX2 - pointSwichesRastX1
Dy2 = pointSwichesRastY2 - pointSwichesRastY1


angl1 = math.atan2(Dy1,Dx1)
angl2 = math.atan2(Dy2,Dx2)
angl =  -(angl1 - angl2)
print(angl1)
print(angl2)
print(angl)

layer = project.mapLayersByName("treangle1")[0]
feat = layer.getFeatures()
suri = "MultiPolygon?crs=epsg:20008&index=yes"
vl = QgsVectorLayer(suri, "treangle2" , "memory")
for feature in feat:
    a = []
    b = []
    for points in feature.geometry().asPolygon()[0]:
        #расстояние от объекта до ключевой точки на векторном изображении
        xRast =  points[0] - pointSwichesVectX1
        yRast =  points[1] - pointSwichesVectY1
        #расстояние от объекта до ключевой точки на растровом изображении
        xNew = pointSwichesRastX1 + xRast
        yNew = pointSwichesRastY1 + yRast
        # поворот координат
        x = ((xNew  - pointSwichesRastX1)* math.cos(angl) - (yNew  -
pointSwichesRastY1)* math.sin(angl) + pointSwichesRastX1)
        y = ((xNew  - pointSwichesRastX1)* math.sin(angl) + (yNew  -
pointSwichesRastY1)* math.cos(angl) + pointSwichesRastY1)
        b.append(QgsPointXY(x, y))
    b.append(b[0])
    a.append(b)
    i += 1
    fet = QgsFeature()
    pr = vl.dataProvider()
    vl.updateExtents()
    fet.setGeometry(QgsGeometry.fromPolygonXY(a))
    pr.addFeatures([fet])
    vl.updateExtents()

    if not vl.isValid():
        print("Layer failed to load!")
    else:
        QgsProject.instance().addMapLayer(vl)
```