

МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное учреждение
высшего образования

**«Российский государственный гуманитарный университет»
(ФГБОУ ВО "РГГУ")**

**ИНСТИТУТ ИНФОРМАЦИОННЫХ НАУК И ТЕХНОЛОГИЙ БЕЗОПАСНОСТИ
ФАКУЛЬТЕТ ИНФОРМАЦИОННЫХ СИСТЕМ И БЕЗОПАСНОСТИ**

Кафедра информационных технологий и систем

УТВЕРЖДАЮ
Заведующий кафедрой ИТС
к.т.н. доцент Роганов А.А.

«__» _____ 2020 г.

Сидоренко Николай Андреевич

ОТЧЕТ ПО ПРАКТИКЕ

студента направления подготовки 09.03.03 «Прикладная информатика»
профиль: «Прикладная информатика в гуманитарной сфере»
(уровень - прикладной бакалавриат)

Вид практики: учебная по получению первичных профессиональных умений и навыков

Сроки прохождения практики: 04.09.2019 – 23.06.2020


Место прохождения практики: МИНОТ, международная лаборатория по проблемам информатики, мехатроники и сенсорики

Руководитель от РГГУ:
зав. кафедрой ИТиС, к.т.н., доцент

_____ А.А. Роганов

«__» _____ 2020 г.

Руководитель от предприятия:
зав. лабораторией

 В. Е. Пряничников

«__» _____ 2020 г.

Москва 2020

Цель работы:

Реализация обработки изображений, изготовление сканирующей головки в т. ч. распознавание пятен с использованием ПЛИС (Python), а также получение практических навыков.

Полученные задачи:

- Освоение базы языка программирования Python;
- Реализация обработки изображения, нахождения центра и смещения центра;
- Проведение сдвига поля зрения т. н. «сканирующей головки», наведение поля зрения на необходимый объект;
- Реализация увеличения изображения для более точного определения центра;
- Определение контуров у изображения;
- Определение контуров у фотоснимка;
- Настройка рабочей камеры
- Настройка управления камерой через веб-интерфейс

Ход работы:

Для поставленной задачи, в первую очередь возникла необходимость изучения языка Python. В качестве среды, была использована Microsoft Visual Studio, а также дополнительно подключена библиотека Python Imaging Library или же PIL, для работы с растровыми изображениями.

Первой задачей было нахождение центра пятна/точки/круга и сдвига этого самого центра, потому для этого были созданы, несколько изображений, с пятном расположенным в разных местах, и для начала работа будет происходить лишь с черно-белым изображением.

Листинг 1 – Основной файл программы определяющий центр

```
from PIL import Image, ImageDraw
import module1 #Импорт модуля, где отдельно прописана функция нахождения центра
img1=Image.open('d:\\imgforpython2\\img1.png') #Открытие изображений
img2=Image.open('d:\\imgforpython2\\img2.png')
img3=Image.open('d:\\imgforpython2\\img3.png')
img4=Image.open('d:\\imgforpython2\\img4.png')
img5=Image.open('d:\\imgforpython2\\img5.png')
imgarr = [img1, img2, img3, img4, img5] #Занечение изображений в массив
x_move = list() #Массив сдвига центра по X
y_move = list() #по Y соответственно
x_pos = 0
y_pos = 0
xx = 0
yy = 0
```

```

for i in range(0, 5): #Через цикл находятся все смещения "пятна"
    x_pos, y_pos = module1.FindSpot(imgarr[i]) #возвращение значений из функции в модуле
    if i == 0:
        xx = x_pos
        yy = y_pos
    x_move.append(x_pos - xx)
    y_move.append(y_pos - yy)
    xx = x_pos
    yy = y_pos
print(x_move) #Вывод массивов смещения координат по X и по Y
print(y_move)

```

Листинг 2 – Модуль module1

```

from PIL import Image, ImageDraw
def FindSpot(img): #объявление метода/функции, в качестве параметра принимается изображение
    width = img.size[0] #Получение ширины изображения
    height = img.size[1] #Получение высоты изображения
    x_list = list() #Массив координат по X
    y_list = list() #Массив координат по Y
    x_pos = 0 #Сюда будут записаны итоговые координаты центра по X
    y_pos = 0 #и по Y соответственно
    count = 0 #Счетчик для определения кол-ва точек
    for i in range(0, width):
        for j in range(0, height):
            pix = img.getpixel((i, j)) #считывания пикселя по опр. координатам
            if (pix <= (100, 100, 100)): #считывание темных пикселей с учетом серых оттенков
                x_list.append(i) #добавление координат в массивы
                y_list.append(j)
                count += 1

    for i in range(0, count): #суммирование всех координат
        x_pos += x_list[i]
        y_pos += y_list[i]
    x_pos = x_pos//count #нахождение среднего значения "центра массы"
    y_pos = y_pos//count
    print("Center of spot: ", x_pos, y_pos) #вывод координат центра
    return x_pos, y_pos

```

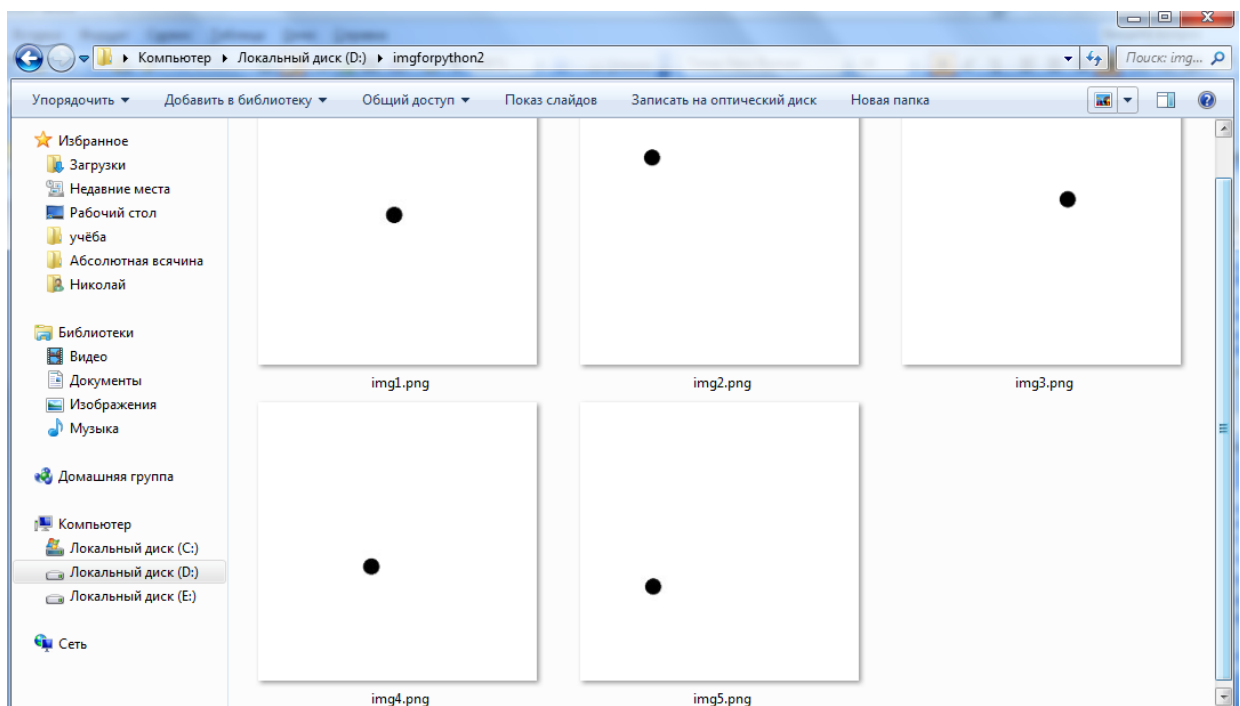


Рис. 1 Изображения содержащие пятно

Программа определяет точные координаты центров пятен, их смещение и выводит их на консоль:

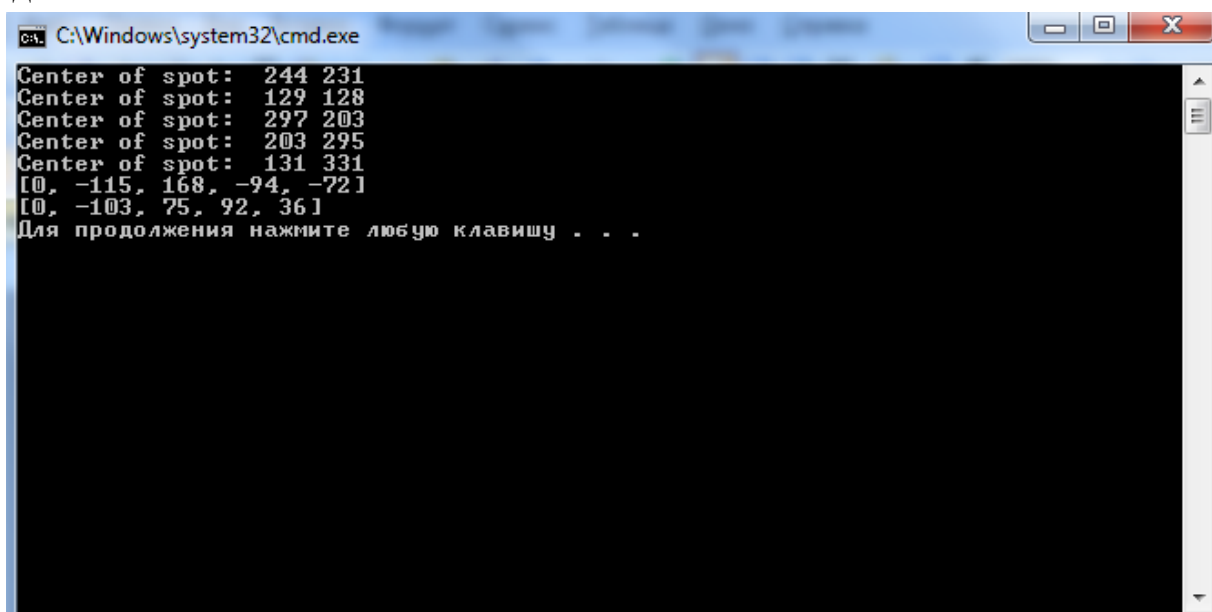


Рис. 2 Вывод центров и смещения

Однако необходимо учитывать возможный сдвиг объекта на изображении. По задумке, у т. н. «сканирующей головки» есть определенное поле зрения, в пределах которого она может воспринимать изображение, и нужно предусмотреть её перемещение по всей области, с целью наведения на объект (в данном случае пятно).

Не находя пятно, поле зрения перемещается по области и в случае нахождения, уже наводится и определяет центр.

Условно поле зрения было установлено на 128x128.

Листинг 3 – Основной файл программы с учетом ограниченного поля зрения

```
import module1
from PIL import Image, ImageDraw #Подключим необходимые библиотеки.
x_pos=0 # Центр
y_pos=0

image1 = Image.open('d:\\BigImage\\Krug1.png') #Открываем изображение.
image2 = Image.open('d:\\BigImage\\Krug2.png')
image3 = Image.open('d:\\BigImage\\Krug3.png')
image4 = Image.open('d:\\BigImage\\Krug4.png')

masIm = [image1,image2,image3,image4] # Массив изображений
draw = ImageDraw.Draw(image1)
x_move = list() # Создание массивов, в котором храниться сдвиг
y_move = list()
width = image1.size[0] #Определяем ширину.
height = image1.size[1] #Определяем высоту.
```

```

print ("width x height ", width,'x',height) # выводим кол-во вертикальных и горизонтальных
пикселей
for i in range (0,4):
    x_pos,y_pos = module1.center(width,height,masIm[i],) # Получение координат центра из модуля
    if i == 0: # Условие на первое изображение
        x_star = x_pos # Запись старой переменной x, если это первое изображение
        y_star = y_pos # Запись старой переменной y, если это первое изображение
        # draw.point((x_pos, y_pos), (255, 255, 255, 255))

    x_move.append(x_pos - x_star) #Добавление элемента массива,равного смещению по x
    y_move.append(y_pos - y_star) #Добавление элемента массива,равного смещению по y
    y_star = y_pos # Запись центра предыдущего изображения
    x_star = x_pos # Запись центра предыдущего изображения
print("Сдвиг: ",x_move,"\n",y_move)
#image1.save("ans.png", "PNG")
del draw

```

Листинг 4 – Модуль module1

```

from PIL import Image, ImageDraw #Подключим необходимые библиотеки.
def center(width,height,masIm):

    pix = masIm.load() #Выгружаем значения пикселей в pix
    flag=True; # Принимает значение Тру, если круг не был обнаружен
    flag2 = False; # Принимает значение Тру, если круг выходит за границы видимости
    x_pos = 0
    y_pos = 0
    count=0
    diap_x = 0 #диапазон по x
    diap_y = 0 #диапазон по y
    while(flag): # Выполняется только при ненахождении круга
        if (diap_x > (width - 128) ): # Не позволяет выйти за границы большого изображения
            diap_x =(width - 128)
        if (diap_y > (height - 128) ):
            diap_y =(height - 128)
        if (diap_x < 0 ): # Не позволяет выйти за границы большого изображения в отрицательную
сторону
            diap_x =0
        if (diap_y <0 ):
            diap_y =0

        for x in range(diap_x,(diap_x+128)): #Прогоняем все изображение через циклы

            for y in range(diap_y,(diap_y + 128)):
                if ( pix[x,y] == (0, 0, 0) or pix[x,y] == (0, 0, 0, 255)): #Суммируется все
количество пикселей по координатам
                    x_pos+=x
                    y_pos+=y
                    if (x == diap_x ): # Если есть черный пиксель у левой рамки
                        diap_x -= 20; # Рамки сдвигаются влево на 20 пикс.
                        flag2= True
                        x_pos=0
                        print("x == diap_x ")
                        #break
                    if (x == (diap_x + 128) ):
                        diap_x += 20;
                        flag2= True
                        x_pos=0
                        print("diap_x + 128")
                        #break
                    y_pos+=y
                    if (y == diap_y ):
                        diap_y -= 20;

```

```

        if (diap_y<0):
            diap_y=0
        flag2= True
        y_pos=0
        print("diap_y")
        #break
    if (y == (diap_y+128) ):
        diap_y += 20;
        flag2= True
        y_pos=0
        print("diap_y+128")
        #break
    if (flag2):
        print("Входит не полностью")
        if (diap_x <0):
            diap_x=0
        x=diap_x
        flag2=False
        if (diap_y <0):
            diap_y=0
        y=diap_y-1

        count+=1 #считается количество черных пикселей
    if count>0 : # Если есть хоть 1 черный пиксель,то считает центр
        x_pos= x_pos//count
        y_pos=y_pos//count
        x_move = x_pos
        y_move = y_pos
        print("center =",x_pos,y_pos)
        flag = False; # При выявлении центра, программа перестает искать что-либо
    else:
        flag = True; # При необноружении центра - поиск продолжается
        print("Kryga net") # Выводится сообщение,что круга в пределах видимости нет
        diap_x+=128; # Смещение вправо,для поиска круга
        if ( diap_x == width): # При достижении верт. границы - переходит на "строку" вниз
            diap_x = 0
            diap_y +=128
            print("Kryga net")
        if ( diap_y+128 == height): # При достижении конца изображения - начинает сначала
            diap_x = 0
            diap_y =0
            print("Ne nashol")
    return x_pos,y_pos

```

```
cmd: C:\Windows\system32\cmd.exe
width x height  500 x 500
Kryga net
Kryga net
center = 301 96
Kryga net
x == diap_x
Входит не полностью
center = 160 70
Kryga net
Kryga net
Kryga net
Kryga net
Kryga net
center = 222 196
Kryga net
Kryga net
center = 349 91
Сдвиг:  [0, -141, 62, 127]
        [0, -26, 126, -105]
Для продолжения нажмите любую клавишу . . .
```

Рис. 3 Вывод программы, определяющий центр и меняющей поле зрения

Для большей точности, возникает необходимость увеличения изображения и нахождения центра с десятикратной точностью. Для этого каждый пиксель повторяется в пределах области 10x10 пикселей, а программа рисует новое изображение с уже увеличенным вариантом пятна.

Листинг 5 – Основной файл программы увеличивающей изображение

```
from PIL import Image, ImageDraw
import module1
img1=Image.open('d:\\imgforpython3\\img4.png')
img2=Image.open('d:\\imgforpython3\\zoomed 2.png')
module1.FindSpot(img1, img2)
```

Листинг 6 – Модуль module1, содержащий основной алгоритм

```
from PIL import Image, ImageDraw
import math
def FindSpot(img1, img2):
    draw = ImageDraw.Draw(img2) #создание инструмента draw для увеличенного изображения
    x_list = list() #список координат по X всех точек пятна
    y_list = list() #список координат по Y всех точек пятна
    width1 = img1.size[0] #ширина изображения
    height1 = img1.size[1] #высота изображения
    width2 = img2.size[0] #ширина изображения
    height2 = img2.size[1] #высота изображения
    x_pos = 0 #координата центра пятна по X
    y_pos = 0 #координата центра пятна по Y
    xmin = 0
    xmax = 0
    ymin = 0
    ymax = 0
    xcount = 0
    ycount = 0
    count = 0 #счетчик всех точек
    for i in range(0, width1):
        for j in range(0, height1):
```

```

    pix = img1.getpixel((i, j)) #в переменную записывается значение пикселя
    if (pix < (250, 250, 250) and xmin == 0): #если пятно черное или содержит оттенки
серого
        xmin = i
    if (pix < (250, 250, 250)):
        xmax = i
    if (pix < (250, 250, 250) and j > ymax):
        ymax = j
        ymin = j
    if (pix < (250, 250, 250) and j < ymin):
        ymin = j
for i in range(xmin, xmax):
    xcount = xcount + 1
    ycount = 0

    for j in range(ymin, ymax):
        pix = img1.getpixel((i, j)) #в переменную записывается значение пикселя
        #if (pix < (250, 250, 250)): #если пятно черное или содержит оттенки серого
        ycount = ycount + 1
        #print(ycount, xcount)
        for k in range(0, 10):
            for p in range(0, 10):
                draw.point((xcount*10+k, ycount*10+p), pix) #рисуются увеличенное в 10 раз
пятно
for i in range(0, width2):
    for j in range(0, height2):
        pix = img2.getpixel((i, j)) #в переменную записывается значение пикселя
        if (pix < (250, 250, 250)):
            x_list.append(i) #в список добавляется координата по X новой найденной точки
            y_list.append(j) #то же по Y, разумеется уже на увелич. изображении
            count += 1
for i in range(0, count):
    x_pos += x_list[i] #сумматор (???) координат по X
    y_pos += y_list[i] #по Y
x_pos = x_pos//count #получается среднее значение по X
y_pos = y_pos//count #по Y
print("Center of spot: ", x_pos, y_pos) #отображение центра пятна
print("Diap of X: ", xmin, xmax) #отображение центра пятна
print("Diap of Y: ", ymin, ymax) #отображение центра пятна
draw.point((x_pos, y_pos), (255, 255, 255)) #обозначение центра белым перекрестием
draw.point((x_pos, y_pos+2), (255, 255, 255))
draw.point((x_pos, y_pos-2), (255, 255, 255))
draw.point((x_pos, y_pos+3), (255, 255, 255))
draw.point((x_pos, y_pos-3), (255, 255, 255))
draw.point((x_pos+2, y_pos), (255, 255, 255))
draw.point((x_pos-2, y_pos), (255, 255, 255))
draw.point((x_pos+3, y_pos), (255, 255, 255))
draw.point((x_pos-3, y_pos), (255, 255, 255))
#img2.save("d:\\imgforpython3\\zoomed.png", "PNG")
img2.show() #открытие изображения

```

Программа выводит на консоль координаты центра пятна, а также открывает увеличенное изображение, отображая центр в виде перекрестия.

Можно просто открыть временно создаваемое изображение, а можно также сохранить на отдельном файле при помощи функции *image.save(*путь к файлу*)*

Результаты работы можно увидеть на рис. 5 – 6.



Рис. 4 Оригинальное изображение перед увеличением

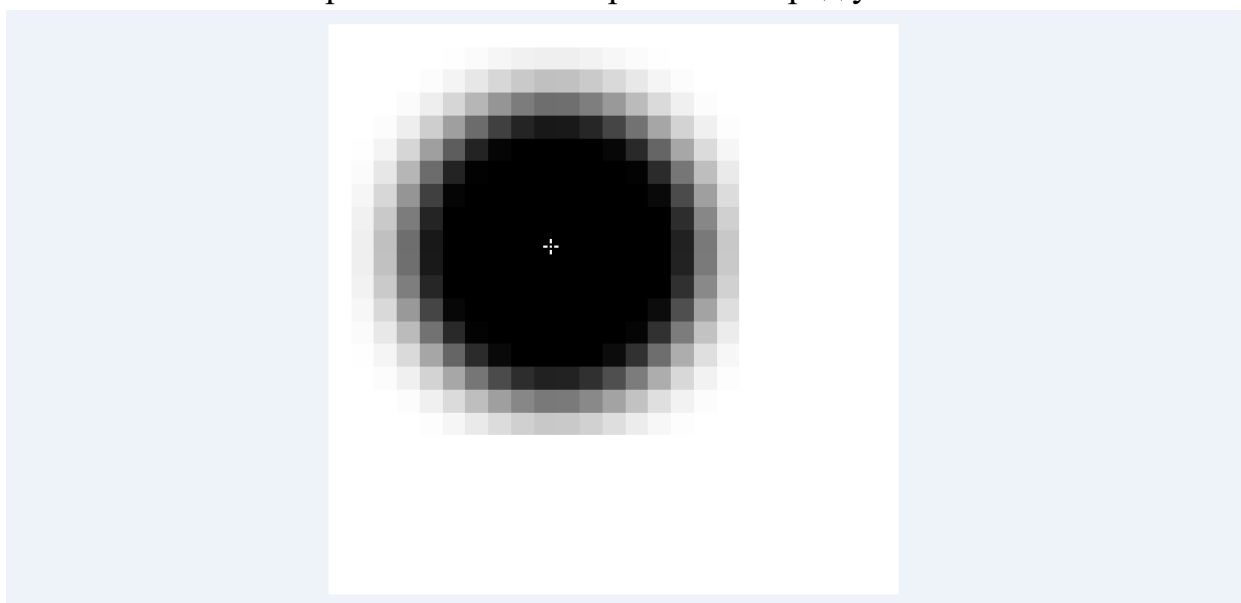


Рис. 5 Увеличенное изображение в 10 раз с найденным центром пятна

Однако работа с пятнами, была лишь начальным этапом перед последующей работой. В ближайшей перспективе была получена задача определения координат, по которым будет двигаться сканирующая головка у лазерного станка (?).

Допустим, есть необходимость вырезать изображенную фигуру на заготовке. Для этого необходима более конкретная обработка изображения. Были порекомендованы различные операторы, например перекрестный оператор Робертса.

В компьютерном зрении – это один из ранних алгоритмов выделения границ, который вычисляет на плоском дискретном изображении сумму квадратов разниц между диагонально смежными пикселями.

Иными словами, величина перепадами G получаемого изображения вычисляется из исходных значений параметра Y в дискретных точках раstra с координатами (x, y) по правилу:

$$G_1 = Y_{x,y} - Y_{x+1,y+1}$$

$$G_2 = Y_{x+1,y} - Y_{x,y+1}$$

$$G = \sqrt{G_1^2 + G_2^2}$$

Однако для корректной работы, изображение должно быть переформатировано в оттенки серого, для того, чтобы RGB значения совпадали, и применение формул стало возможным. Также полученное значение G нужно отдельно округлить, поскольку цветное значение, представляющее из себя «кортеж» из трёх байтов, например 255, 255, 255 (белый цвет, имеющий максимальные значения байтов), не может принимать значение с плавающей точкой.

Следующий блок кода, преобразовывает каждый пиксель в оттенок серого, усредняя его RGB-значения.

Листинг 7 – Блок кода, преобразующий цвета на изображении в оттенки серого

```
for i in range(0, width3):
    for j in range(0, height3):
        pix = img3.getpixel((i, j))
        r = pix[0]
        g = pix[1]
        b = pix[2]
        S = (r + g + b) // 3
        draw.point((i, j), (S, S, S))
```

Далее уже реализуется оператор. Если говорить простым языком, сперва, в переменные G_1 и G_2 записываются разности диаметрально противоположных пикселей. Далее в G сохраняется корень суммы квадратов этих разностей, который впоследствии также округляется при помощи метода *floor*. Для функций вроде *floor* и *sqrt* (квадратный корень), необходимо в коде импортировать библиотеку *math*.

Листинг 8 – Блок кода с реализацией оператора Робертса

```
for i in range(0, width3 - 1):
    for j in range(0, height3 - 1):
        pix00 = img3.getpixel((i, j))
        pix01 = img3.getpixel((i, j+1))
        pix10 = img3.getpixel((i+1, j+1))
        pix11 = img3.getpixel((i+1, j))
        Y00 = pix00[0]
        Y01 = pix01[0]
        Y10 = pix10[0]
        Y11 = pix11[0]
        G1 = Y00 - Y11
        G2 = Y10 - Y01
```

```

G = math.floor(math.sqrt((G1**2)+(G2**2)))
if (G < 15):
    draw2.point((i, j), (0, 0, 0))
if (G >= 15):
    draw2.point((i, j), (G+20, G+20, G+20))
if (G >= 30 and G < 205):
    draw2.point((i, j), (G+50, G+50, G+50))

```

Полученное значение G записывается в пиксель с индексами $[0][0]$, в цикле это текущие $[i][j]$, но чтобы более четко выделить границы, можно повысить их яркость, если G выше определенного значения, или наоборот не учитывать, закрашивая в черный, если G слишком мал. Результат работы оператора, можно увидеть на рис. 6 – 7.

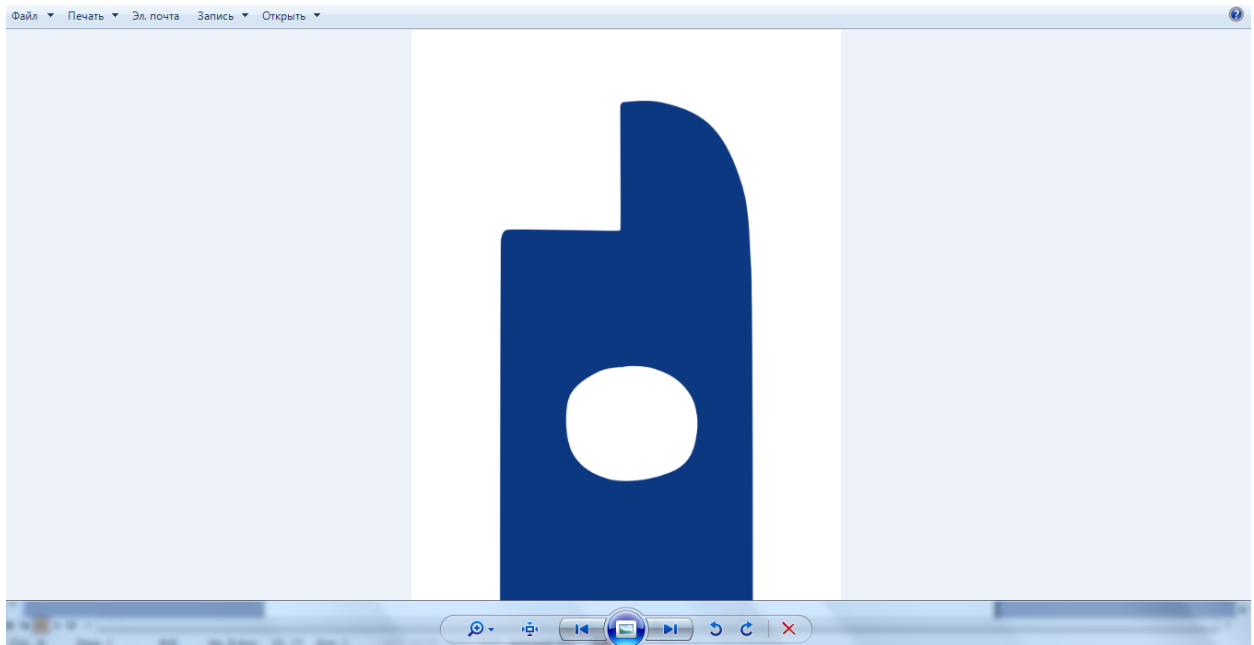


Рис. 6 Произвольная фигура

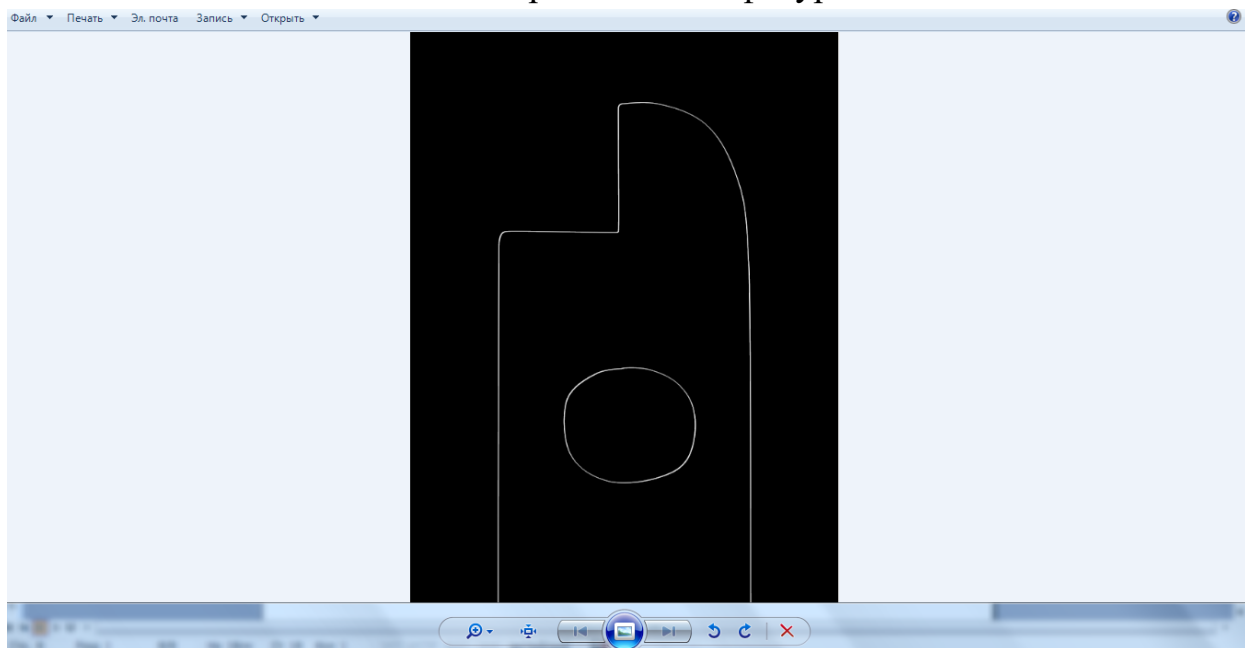


Рис. 7 Выделенный контур произвольной фигуры

После обработки изображения со случайной формой, получается достаточно точный контур.

Однако в данном случае использовалось созданное в графическом редакторе изображение. Есть необходимость проверить работу алгоритма с фотоснимком, в котором дополнительно появляются такие моменты как освещение, шероховатости, «шум». Использованная ранее форма была распечатана на принтере и сфотографирована.

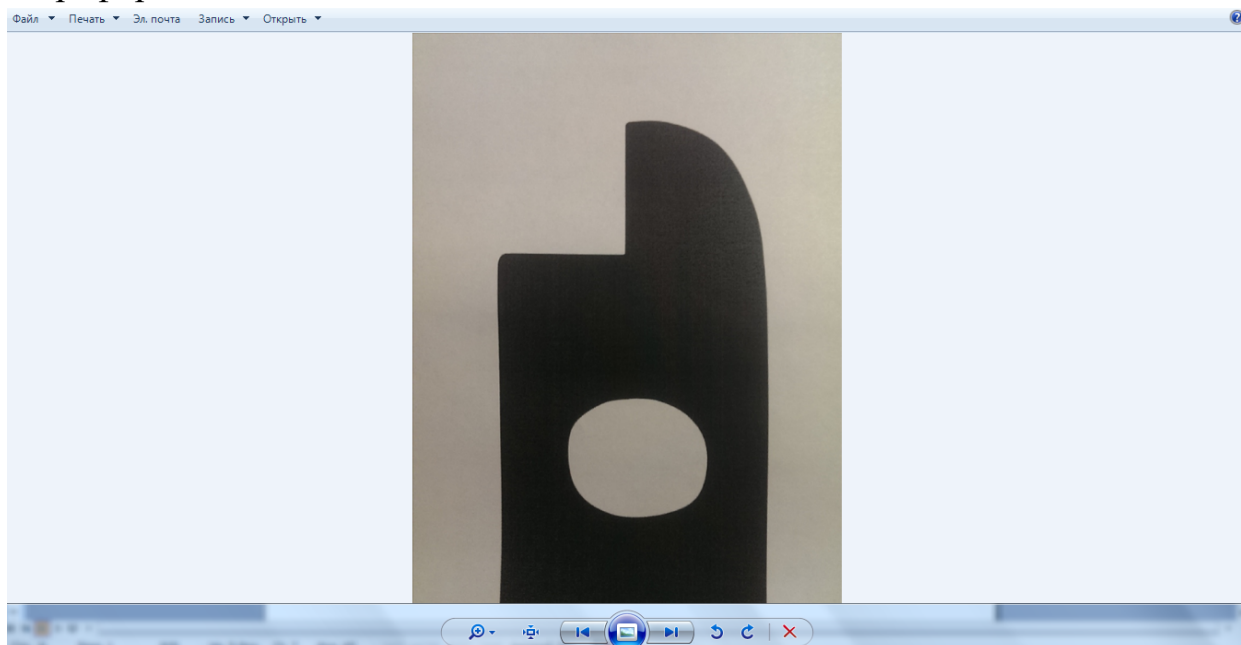


Рис. 8 Фото произвольной фигуры

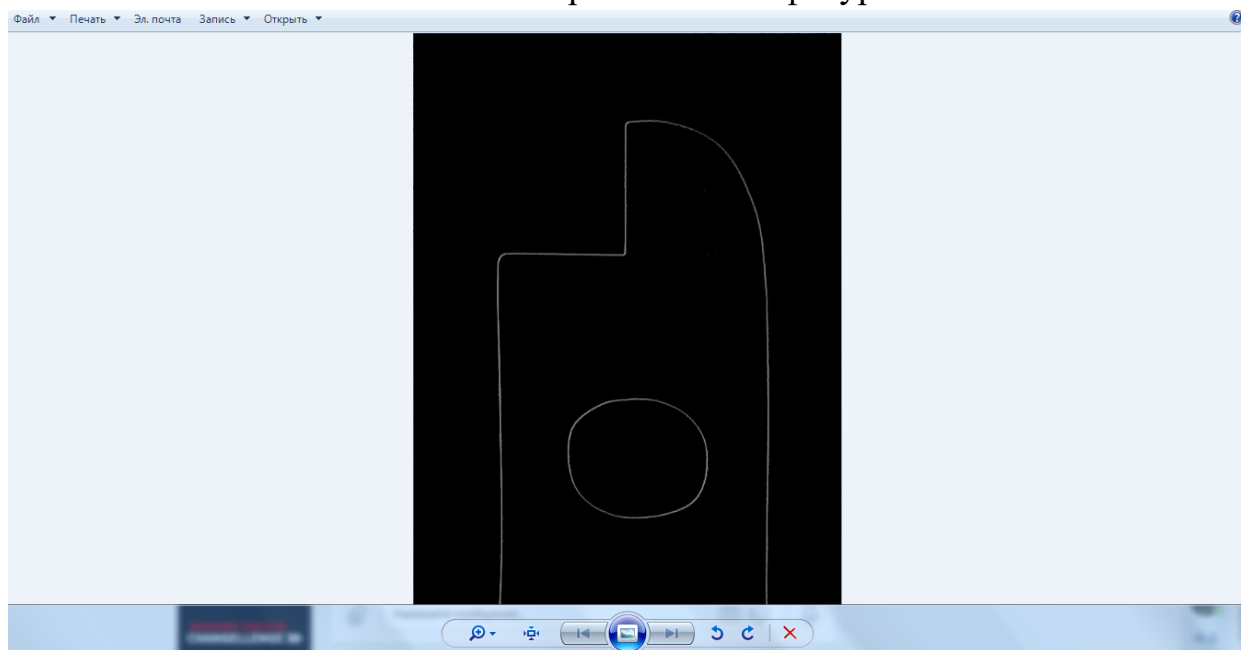


Рис. 9 Выделенный контур фигуры на фото

Алгоритм сработал. По полученному контуру в дальнейшем при

необходимости можно будет, например, создавать координатную последовательность, по которой будет происходить движение камеры.

Далее по ходу работы, возникла новая задача, связанная с сканирующей головки, а потому возникла необходимость более подробно изучить работу камеры. В устройстве установлена камера STC-MC202USB. Для того чтобы понять, какое будет получено изображение, потребовалось установить драйвера для камеры, а также программу Sentech Viewing Software, которая через USB-подключение в режиме реального времени получает изображение от камеры. И с самого начала возник ряд проблем связанных с настройками камеры, её фокусировкой, а также освещением. На рисунке 10 отображено первое полученное с камеры изображение, но кроме сильной размытости, есть также проблема с яркостью.

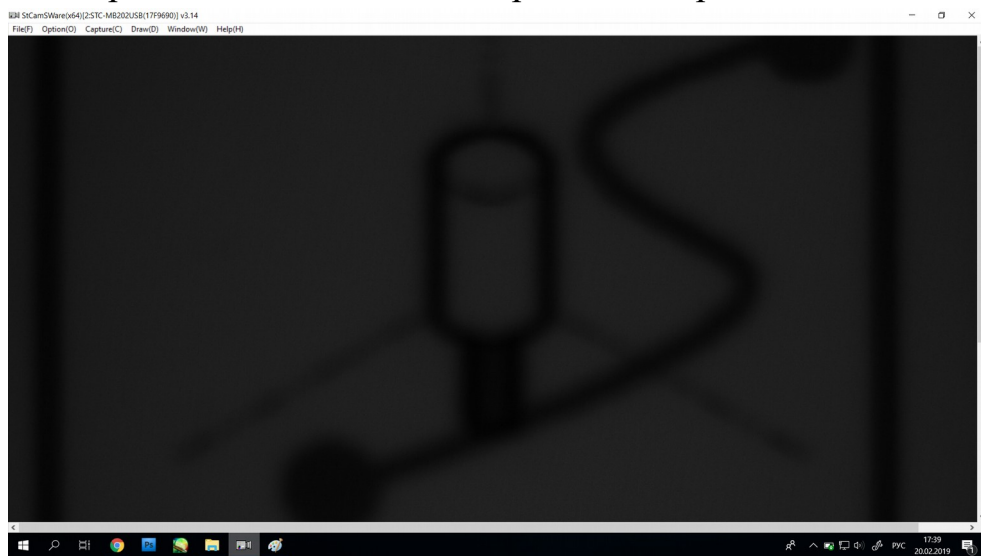


Рис. 10 Первое полученное с камеры изображение

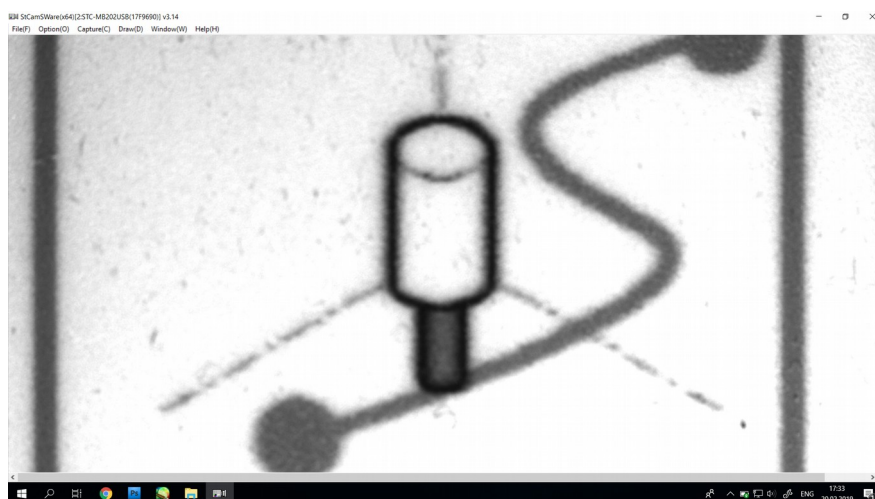


Рис. 11 Изображение с добавлением яркого освещения

Добавление яркого освещения, как видно на рис. 11, улучшило видимость, но решить проблему размытости не помогло, кроме того высокая светочувствительность ухудшает качество получаемой картинки. Потому было решено понизить светочувствительность до минимума, но обеспечить рабочую поверхность достаточно ярким светом, как показано на рис. 12.

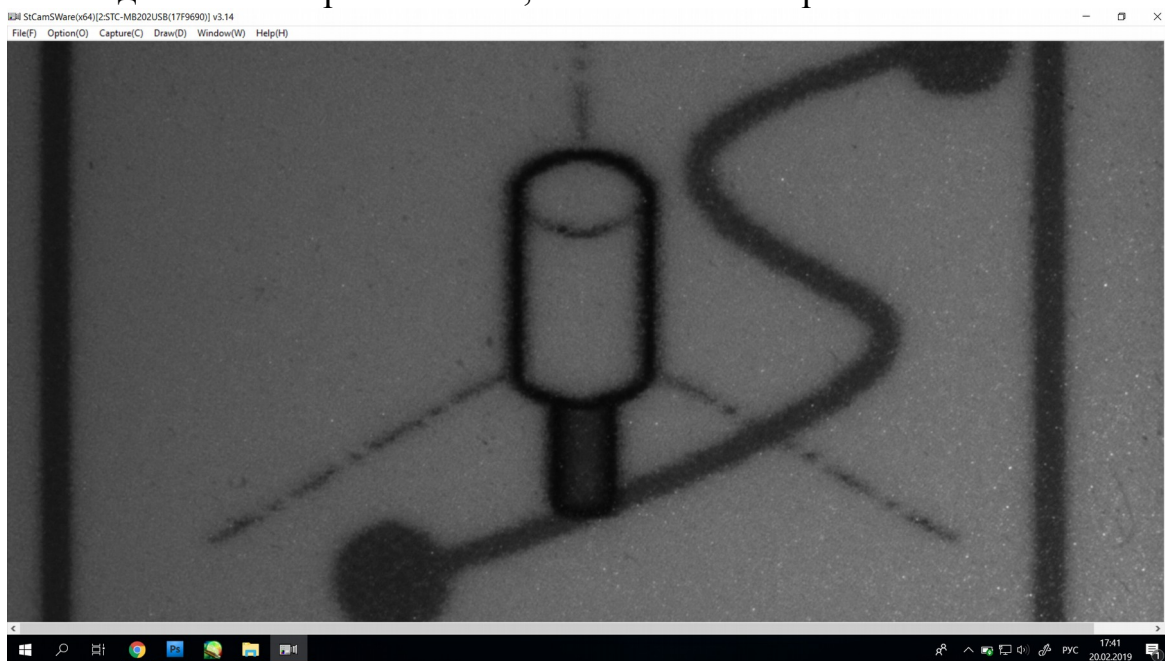


Рис. 12 Освещенное изображение с низкой светочувствительностью камеры

Таким образом, было получено наиболее качественное изображение из возможных. На первый взгляд может показаться, что качество всё еще оставляет желать лучшего, однако тут еще есть проблема самого рабочего трафарета. Подложив под камеру другую картинку, получилось удостовериться в максимальной четкости изображения, как показано на рис. 13.

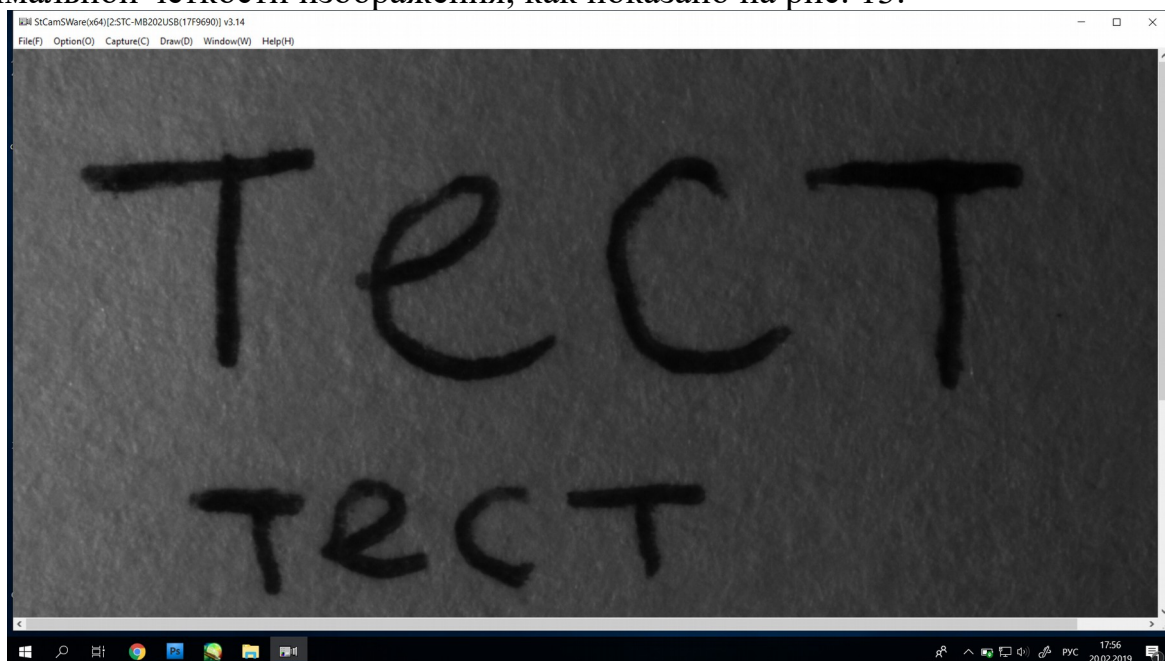


Рис. 13 Работа настроенной камеры с другим изображением

Дальнейшая задача заключается в освоении формата обмена данных JSON. Это достаточно легкий для освоения формат, библиотека которого присутствует в python. Такой файл может содержать структуры или массивы с какими-либо данными, и программа может их при необходимости считать. По задумке команды, было решено, что программа будет получать при помощи запроса код такого формата, содержащий ссылки на изображения, которыми в могут быть, например кадры с камеры. Пока, камера еще не настроена под работу с API, поэтому был разработан специальный трафарет изображенный на рисунке 14.

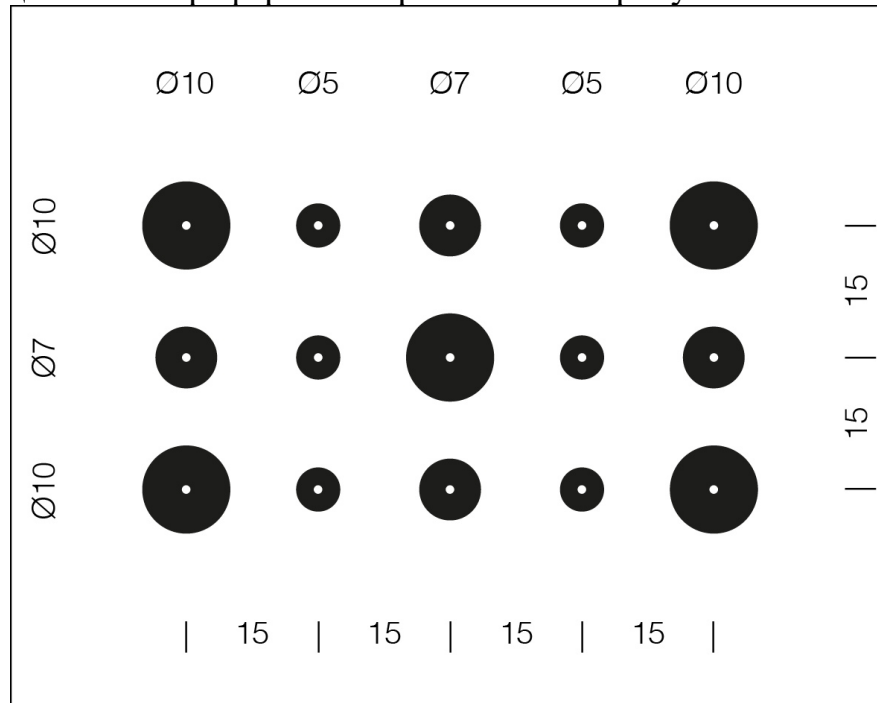


Рис. 14 Трафарет для рабочей камеры

Ссылка на скачивание этого трафарета, для примера был размещена на сервере в вышеупомянутом формате. Код получения изображения из структуры JSON-формата изображен далее.

Листинг 9 – Чтение формата JSON

```
import json
import requests
import webbrowser

response = requests.get("http://ctpo.sensorika.info:57264/api/stream/d3336817-8788-4f1f-b916-62bd319c4acf/frames/8")
#try to get json from web
data = json.loads(response.text) #save json to variable

type(data) #parse to string
print(data[1]) #print first structure from array

for i in range(4):
    print(i+1, "-", data[i])
print("What url do you want to open? For example the 1st is our drawn plan of work")
num = input(" Write the number: ")

webbrowser.open(data[int(num)-1]) #open one of url in default browser
```


Полученное изображение можно, к примеру, открыть в браузере по умолчанию, при помощи библиотеки webbrowser, как показано на рисунке 15.

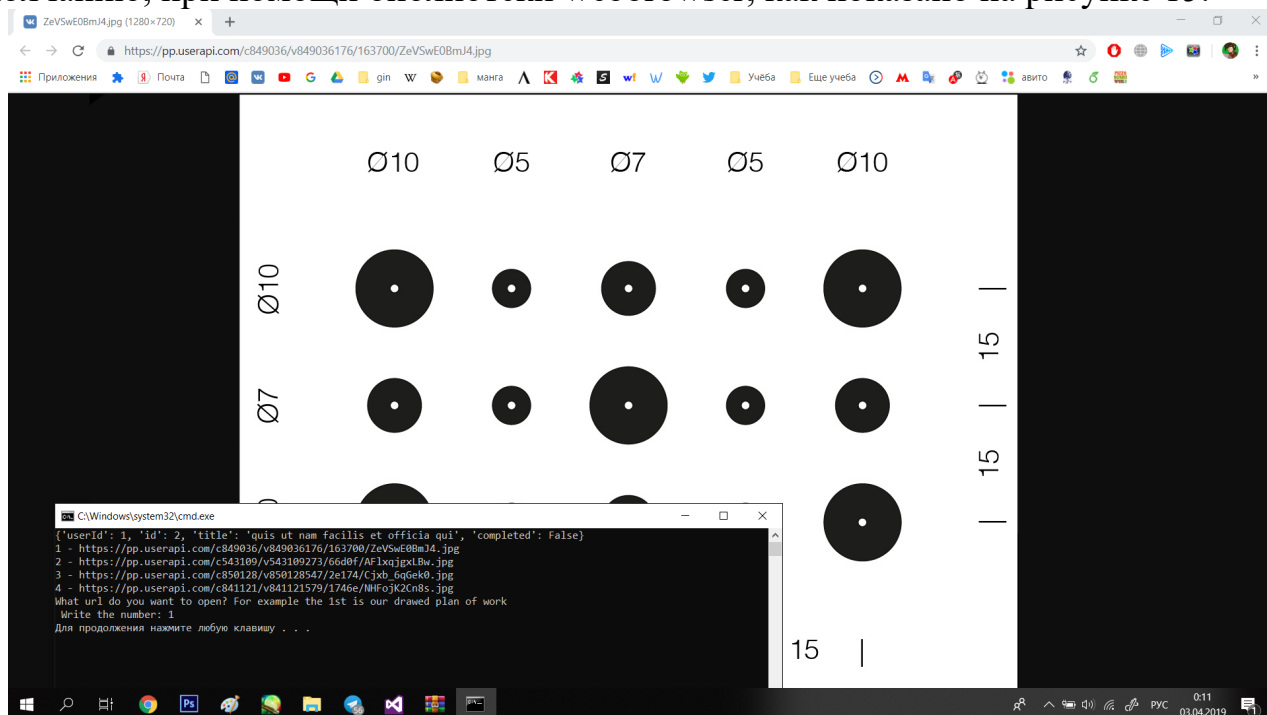


Рис. 15 Получение трафарета по ссылке

После того, как был распечатан трафарет, появилась возможность взаимодействия камеры с ним. Путем прямого подключения к серверу по ip-адресу, был получен доступ к веб-интерфейсу для управления движения камерой.

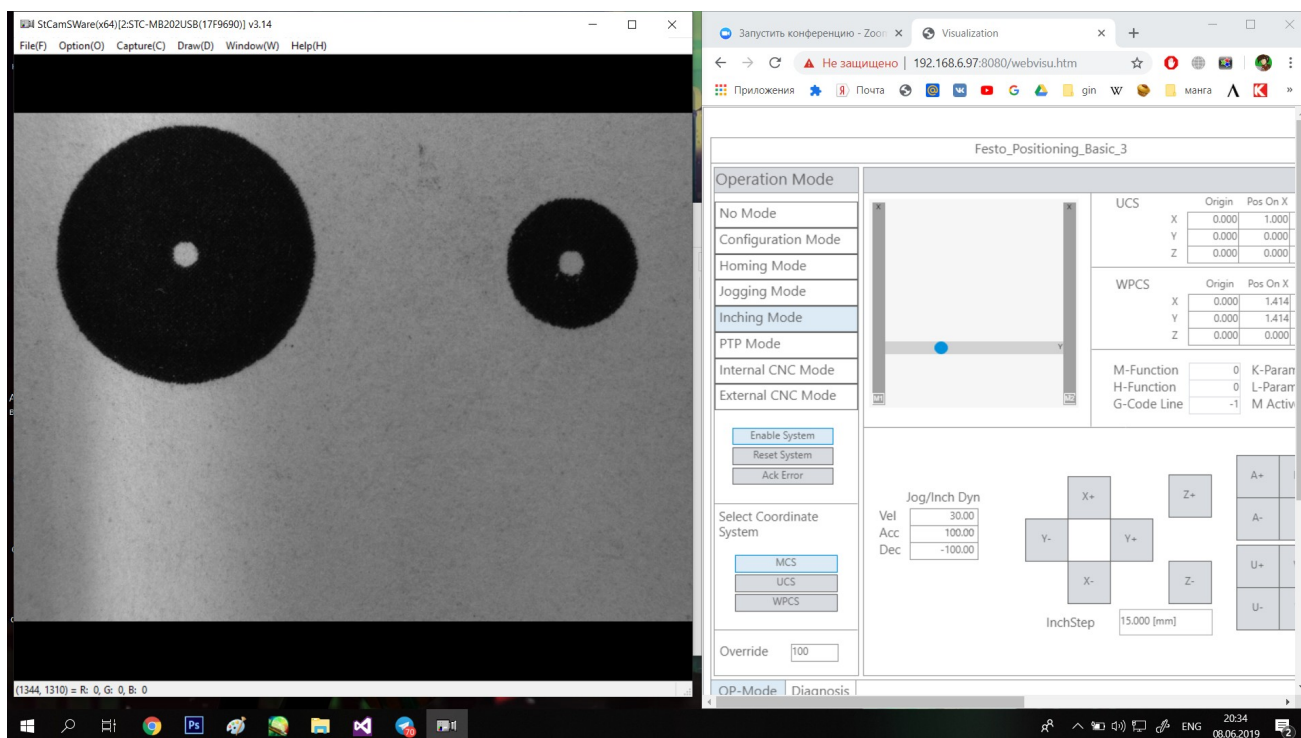


Рис. 16 Управление рабочей камерой

Далее нужно решить вопрос удаленного подключения.

В последние годы REST (REpresentational State Transfer) стала стандартной архитектурой при дизайне веб-сервисов и веб-API.

Характеристика системы REST определяется шестью правилами дизайна:

- Клиент-Сервер: Должно быть разделение между сервером, который предлагает сервис и клиентом, который использует ее.
- Stateless: Каждый запрос от клиента должен содержать всю информацию, необходимую серверу для выполнения запроса. Другими словами, сервер не обязан сохранять информацию о состоянии клиента.
- Кэширование: В каждом запросе клиента должно явно содержаться указание о возможности кэширования ответа и получения ответа из существующего кэша.
- Уровневая система: Клиент может взаимодействовать не напрямую с сервером, а с произвольным количеством промежуточных узлов. При этом клиент может не знать о существовании промежуточных узлов, за исключением случаев передачи конфиденциальной информации.
- Унификация: Унифицированный программный интерфейс сервера.
- Код по запросу: Сервера могут поставлять исполняемый код или скрипты для выполнения их на стороне клиентов.

Архитектура REST разработана чтобы соответствовать протоколу HTTP используемому в сети Интернет.

Центральное место в концепции RESTful веб-сервисов это понятие ресурсов. Ресурсы представлены URI. Клиенты отправляют запросы к этим URI используя методы представленные протоколом HTTP, и, возможно, изменяют состояние этих ресурсов.

Методы HTTP спроектированы для воздействия на ресурс стандартным способом показаны в табл. 1.

Таблица 1 – методы http

Метод HTTP	Действие	Пример
GET	Получить информацию о ресурсе	example.com/api/orders (получить список заказов)
GET	Получить информацию о ресурсе	example.com/api/orders/123 (получить заказ #123)
POST	Создать новый ресурс	example.com/api/orders (создать новый заказ из данных переданных с запросом)
PUT	Обновить ресурс	example.com/api/orders/123

		(обновить заказ #123 данными переданными с запросом)
DELETE	Удалить ресурс	example.com/api/orders/123 (удалить заказ #123)

Дизайн REST не дает рекомендаций каким конкретно должен быть формат данных передаваемых с запросами. Данные переданные в теле запроса могут быть JSON blob, или с помощью аргументов в URL.

Поскольку рабочий компьютер в лаборатории имеет Unix-архитектуру, было решено использовать Unix-подобную операционную систему. Это означает, что они будут работать на Linux, MacOS X и даже на Windows, если использовать Cygwin.

Команды будут несколько отличаться, если использовать нативную версию Python для Windows.

Для начала нужно установить Flask в виртуальном окружении.

Листинг 10 – установка Flask

```
$ mkdir todo-api
$ cd todo-api
$ virtualenv flask
New python executable in flask/bin/python
Installing setuptools.....done.
Installing pip.....done.
$ flask/bin/pip install flask
```

Теперь, когда Flask установлен можно попробовать создать простое веб приложение, для этого нужно будет поместить следующий код в app.py:

Листинг 11

```
#!/flask/bin/python from flask
import Flask app = Flask(__name__) @app.route('/')
def index():
    return "Hello, World!"
if __name__ == '__main__':
    app.run(debug=True)
```

Чтобы запустить приложение, нужно запустить app.py:

Листинг 12

```
$ chmod a+x app.py
$ ./app.py
* Running on http://127.0.0.1:5000/
* Restarting with reloader
```

Теперь можно будет уже запустить веб-браузер и набрать <http://localhost:5000> чтобы увидеть наше маленькое приложение в действии.

Клиенты разрабатываемого веб-сервиса будут просить сервис добавлять, удалять и отправлять ссылки на кадры h-ganry, поэтому нужен простой способ хранить ссылки. Очевидный способ сделать это — сделать небольшую базу данных, но, для данной задачи вместо базы данных было решено хранить список наших ссылок в памяти. Это сработает, только если работать с сервером в один поток и в один процесс.

Хоть для development-сервера это нормально, то для production-сервера это будет очень плохой идеей и будет лучше подумать об использовании базы данных.

Листинг 13 – первая точка входа в сервис

```
#!/flask/bin/python from flask
import Flask, jsonify
app = Flask(__name__)
tasks = [
    {
        'id': 1,
        'title': u'1',
        'description': u'
https://sun9-4.userapi.com/c858228/v858228454/68245/gxrs1DKRc4U.jpg',
        'done': False
    },
    {
        'id': 2,
        'title': u'2',
        'description': u'
https://sun9-57.userapi.com/c855528/v855528126/207c4a/UdgUokvHQM.jpg',
        'done': False
    }
]

@app.route('http://ctpo.sensorika.info:57264/api/stream/d3336817-8788-4f1f-b916-
62bd319c4acf/frames/8', methods=['GET'])

def get_tasks():
    return jsonify({'tasks': tasks})
if __name__ == '__main__': app.run(debug=True)
```

Теперь, вместо того, чтобы использовать точку входа index, у нас теперь есть функция get_tasks связанная с URI /todo/api/v1.0/tasks, для HTTP метода GET.

Вместо текста функция отдает JSON, в который Flask с помощью метода jsonify кодирует нашу структуру данных.

Использование веб-браузера, для тестирования веб-сервиса, не самая лучшая идея, т.к. с помощью веб-браузера не так просто генерировать все типы HTTP-запросов. Вместо этого лучше использовать curl (<https://curl.haxx.se/>)

cURL — (распространяемая по лицензии MIT), кроссплатформенная служебная программа командной строки, позволяющая взаимодействовать с множеством различных серверов по множеству различных протоколов с синтаксисом URL.

Запустить веб-сервис можно тем же самым путем, как и демонстрационное приложение, запустив `app.py`.

Листинг 14 – запуск сервиса

```
$ curl -i http://ctpo.sensorika.info:57264/api/stream/d3336817-8788-4f1f-b916-62bd319c4acf/frames/8
HTTP/1.0 200 OK
Content-Type: application/json
Content-Length: 294
Server: Werkzeug/0.8.3 Python/2.7.3
Date: Mon, 20 May 2020 04:53:53 GMT
```

```
tasks = [
    {
        'id': 1,
        'title': u'1',
        'description': u'
https://sun9-4.userapi.com/c858228/v858228454/68245/gxrs1DKRc4U.jpg',
        'done': False
    },
    {
        'id': 2,
        'title': u'2',
        'description': u'
https://sun9-57.userapi.com/c855528/v855528126/207c4a/UdgUokvHQJM.jpg',
        'done': False
    }
]
```

Были получены ссылки на изображения при помощи RESTful API.

Поскольку данное подключение является относительно простым, его можно также проверить и в браузере.

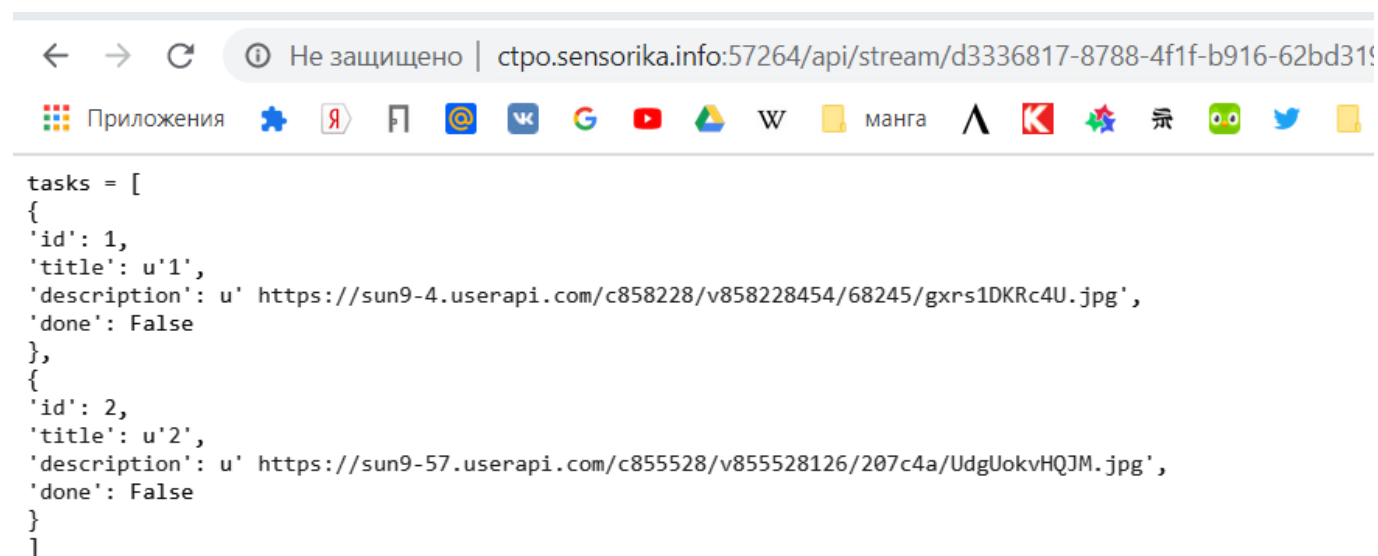


Рис. 17 реализация сервиса в браузере

Простым скриптом на python можно открыть полученный json-текст в качестве структуры и скачать файлы полученные по ссылкам (рис. 18.1 – 18.2).

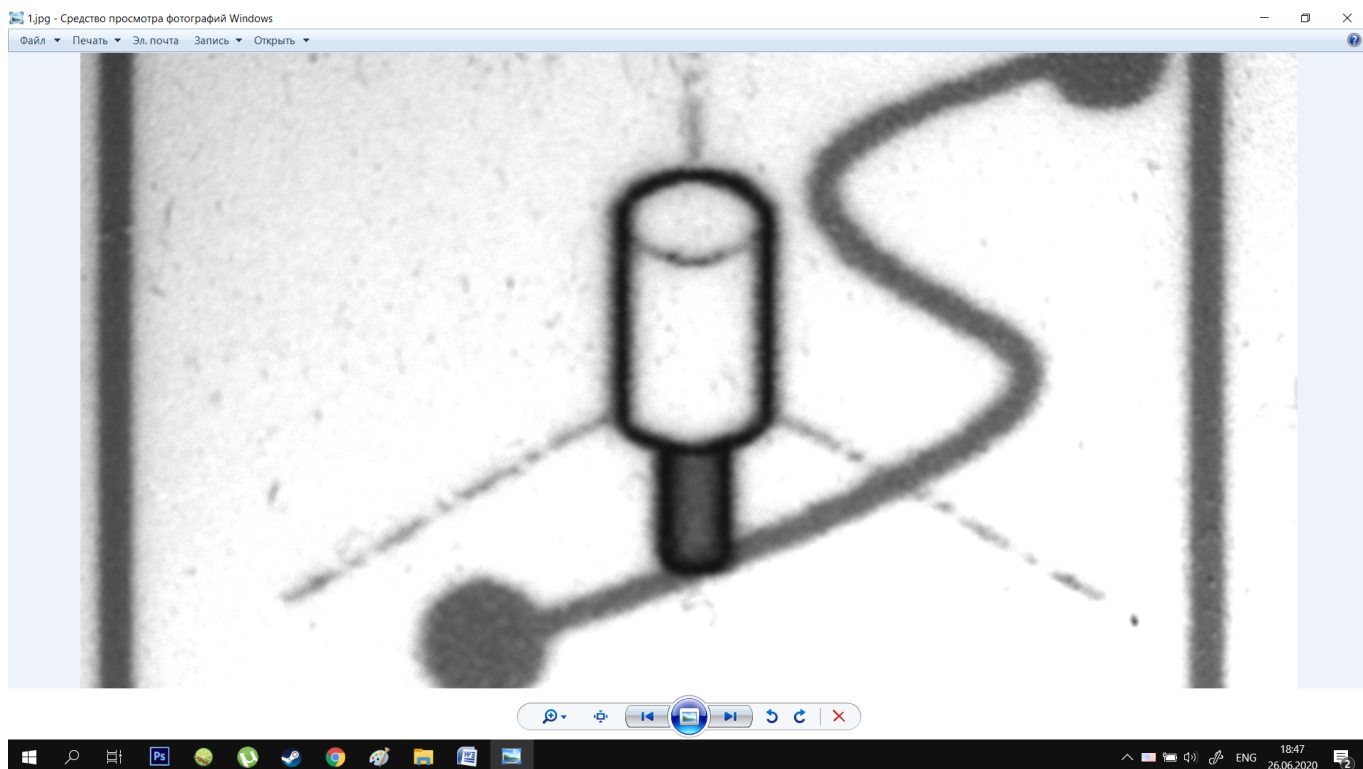


Рис. 18.1 изображение «1»

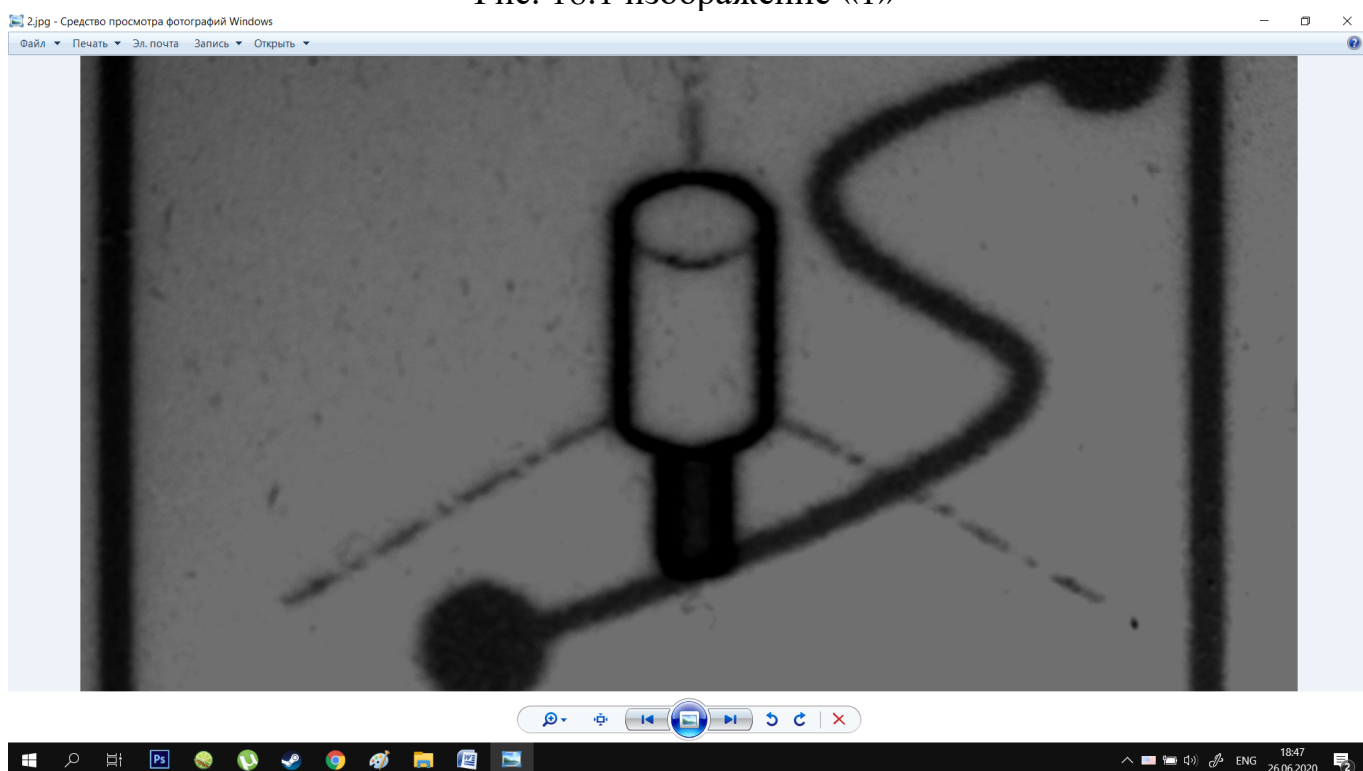


Рис. 18.2 изображение «2»

Вывод

По ходу прохождения практики, были получены некоторые базовые навыки работы в области разработки программного обеспечения. В первую очередь было уделено внимание изучению языка Python, его операторов и библиотек, освоению обработки изображений и реализации алгоритмов нахождения центра, сдвига поля зрения и нахождения объекта, построения контура.

На базе этого в дальнейшем написанные шаблоны можно будет объединить в единую программу и совершенствовать по мере возникновения новых промежуточных задач.

Кроме того были получены знания в области стандартов обмена данными, взаимодействия с сервером, освоено управление передвижением рабочей камеры робота через веб-интерфейс.

Характеристика¹

на студента 3 курса факультета информационных систем и безопасности
Российского государственного гуманитарного университета
Сидоренко Николая Андреевича

Сидоренко Н. А. проходил учебную практику в МИНОТ, международной лаборатории по проблемам информатики, мехатроники и сенсорики на должности «практикант». За время прохождения практики обучающийся ознакомился с: программированием на языке Python, в т.ч. с основными операторами и некоторыми библиотеками, а также алгоритмами обработки изображений, выполнял разработку алгоритма калибровки изображения полученного с устройства, настраивал камеру робота, участвовал в коллективной дискуссии проекта.

За время прохождения практики Сидоренко Н. А. зарекомендовал себя как коммуникативный и способный к обучению и достаточно ответственный практикант. Продемонстрировал умение вести диалог и излагать свои мысли, проявил приемлемую дисциплину и лояльность к организации.

Оценка за прохождение практики – «_отлично_» .

Руководитель практики

Роганов А. А., зав. кафедрой
ИТиС, к.т.н., доцент

(дата)

(подпись)

Руководитель от предприятия



Пряничников В. Е.,
зав. Лабораторией, д.т.н.

(дата)

(подпись)

¹ Оформляется либо на бланке организации, либо заверяется печатью.

ФГБОУ ВО РГГУ
Институт информационных наук и технологий безопасности
Факультет информационных систем и безопасности
Кафедра информационных технологий и систем

УТВЕРЖДАЮ
Заведующий кафедрой ИТС
к.т.н. доцент Роганов А.А.

«__» _____ 20__ г.

ЗАДАНИЕ на практику

Студенту Сидоренко Николаю Андреевичу 3 курса очной формы обучения, направление подготовки 09.03.03 Прикладная информатика, профиль "Прикладная информатика в гуманитарной сфере"

Вид практики: учебная по получению первичных профессиональных умений и навыков

Сроки прохождения практики: 04.09.2019 – 23.06.2020

Место прохождения практики: МИНОТ, международная лаборатория по проблемам информатики, мехатроники и сенсорики

Перечень вопросов, подлежащих рассмотрению:

- Освоение базы языка программирования Python;
- Реализация обработки изображения, нахождения центра и смещения центра;
- Проведение сдвига поля зрения т. н. «сканирующей головки», наведение поля зрения на необходимый объект;
- Реализация увеличения изображения для более точного определения центра;
- Определение контуров у изображения;
- Определение контуров у фотоснимка;
- Настройка рабочей камеры
- Настройка управления камерой через веб-интерфейс
- Удаленное подключение через REST API и Flask

График прохождения практики

Дата (даты)	Раздел практики	Отметка о выполнении
04.09.2019	Ознакомление с приказом и заданием	
09.09.2019	Сбор всех отчетов с прошлого курса, восстановление программных файлов	
16.09.2019	Переустановка microsoft visual studio, восстановление программных файлов	
20.09.2019	Посещение лаборатории	
14.10.2019	Посещение лаборатории, работа над восстановлением сервера на рабочем компьютере	
15.10.2019	Работа над удаленным подключением к серверу	
30.10.2019	Сборка старой программы	
01.11.2019	Разработка трафарета для работы с камерой	
06.11.2019	Повторное изучение python, закрепление основных функций	
07.11.2019	Написание кода	
13.11.2019	Написание кода	
15.11.2019	Восстановление драйверов камеры	
29.11.2019	Настройка программы камеры	
13.12.2019	Посещение лаборатории, попытка управления камерой	
25.12.2019	Настройка программы камеры	
27.12.2019	Посещение лаборатории	
31.12.2019	Посещение лаборатории	
10.01.2019	Конференция в zoom	
13.01.2019	Уточнение критериев необходимых для трафарета с которым будет взаимодействовать рабочая камера	
07.02.2020	Восстановление сервера для получения ссылок на скачивание изображений записанных в формате JSON	
11.02.2020	Получение доступа к удаленному управлению камерой через веб-интерфейс	
14.02.2020	Конференция в zoom, обсуждение работ	
18.02.2020	Написание кода	
21.02.2020	Написание кода	
25.02.2020	Изучение материала по удаленному подключению к серверу при помощи REST API	
03.03.2020	Изучение стандартов и протоколов TCP/IP	
06.03.2020	Посещение лаборатории	
10.03.2020	Посещение лаборатории	
20.03.2020	Переход на удаленное обучение	
24.03.2020	Внеплановые выходные	
25.03.2020	Внеплановые выходные	
07.04.2020	Конференция в zoom, обсуждение плана с учетом	

	карантина	
11.04.2020	Удаленная работа с подключением к серверу через REST API	
25.04.2020	Разработка калибровки изображения	
19.06.2020	Калибровка изображения	
22.06.2020	Финальная подготовка отчета	
23.06.2020	Проверка отчета	

Руководитель практики: _____

(должность, ученая степень, ученое звание, фамилии и инициалы)

(подпись)

(расшифровка подписи)

Руководитель от предприятия: _____ д.т.н., снс Пряничников В.Е. _____

(должность, ученая степень, ученое звание, фамилии и инициалы)



(подпись)

(расшифровка подписи)

« » _____ 20__ г.

Задание получено:

(подпись студента)

(расшифровка подписи)