



Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Московский государственный технический университет
имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

ФАКУЛЬТЕТ ИНФОРМАТИКА И СИСТЕМЫ УПРАВЛЕНИЯ

КАФЕДРА КОМПЬЮТЕРНЫЕ СИСТЕМЫ И СЕТИ (ИУ6)

НАПРАВЛЕНИЕ ПОДГОТОВКИ 09.03.01 Информатика и вычислительная техника

БАКАЛАВРСКАЯ ПРОГРАММА 09.03.01_03 Вычислительные машины, комплексы,
системы и сети

ОТЧЕТ ПО ПРОИЗВОДСТВЕННОЙ ПРАКТИКЕ

Тип практики Технологическая практика

Название АО «РТСофт»
предприятия

Студент ИУ6-42Б

(Подпись, дата)

С.В. Астахов

(И.О. Фамилия)

Руководитель практики

(Подпись, дата)

Е.В. Смирнова

(И.О. Фамилия)

Оценка _____

2021 г.

Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего образования
«Московский государственный технический университет имени Н.Э. Баумана
(национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

УТВЕРЖДАЮ

Заведующий кафедрой ИУ6

А.В. Пролетарский
« 01 » « 07 » 2021 г.

З А Д А Н И Е
на производственную практику

по теме Разработка системного программного обеспечения для
интеллектуальных IoT-устройств

Студент группы ИУ6-42Б

Астахов Сергей Викторович
(Фамилия, имя, отчество)

Направление подготовки 09.03.01 Информатика и вычислительная техника

Бакалаврская программа 09.03.01_03 Вычислительные машины, комплексы, системы и сети

Тип практики Технологическая практика

Название предприятия АО «РТСофт»

Техническое задание Разработать конвертер сообщений Kafka в формат OPC UA. Исходные данные брать с
эмулируемых или реальных датчиков. Визуализировать передаваемые значения.

Оформление отчета по практике:

Отчет на 15-25 листах формата А4.

Перечень графического (иллюстративного) материала (чертежи, плакаты, слайды и т.п.) нет

Дата выдачи задания « 01 » июля 2021 г.

Руководитель практики

Студент

(Подпись, дата)

(Подпись, дата)

Е.В. Смирнова

(И.О. Фамилия)

С.В. Астахов

(И.О. Фамилия)

Примечание: Задание оформляется в двух экземплярах.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ.....	4
ОСНОВНАЯ ЧАСТЬ.....	5
1 Ознакомительные задания	5
1.1 Udev правила.....	5
1.2 Драйверы в Linux.....	6
1.3 Контейнеризация.....	9
1.4 Компьютерное зрение.....	11
1.5 Сетевой протокол MQTT	14
1.6 Apache Kafka.....	18
1.7 TSDB - Базы данных временных рядов.....	20
1.8 REST и http-запросы.....	23
2 Индивидуальное задание.....	27
ЗАКЛЮЧЕНИЕ	31
СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ.....	32

ВВЕДЕНИЕ

Цель производственной практики: изучение механизмов работы с внешними устройствами в ОС семейства Linux, библиотеки алгоритмов компьютерного зрения OpenCV, овладение навыками работы с элементами IoT-инфраструктуры.

Задачи производственной практики:

- Получить знания о работе драйверов и udev правил в ОС семейства Linux
- Получить базовые навыки работы с docker контейнерами
- Получить базовые знания об алгоритмах компьютерного зрения
- Получить базовые навыки работы с сообщениями в сетях типа “интернет вещей”

ОСНОВНАЯ ЧАСТЬ

1 Ознакомительные задания

Перед выполнением итогового индивидуального задания было необходимо выполнить ряд ознакомительных задания с целью освоения базовых навыков работы с IoT-устройствами.

1.1 Udev правила

Udev (userpace / dev) – подсистема Linux для динамического обнаружения и управления устройствами.

Udev запускается как демон и принимает события uevents от ядра, которые генерируются при инициализации или удалении устройства из системы. Задаваемые пользователем (системой) правила сверяются со свойствами события и соответствующего устройства, и совпавшее правило может назвать и создать соответствующий файл устройств, а также выполнить другие программы для инициализации и конфигурации устройства.

Задание: Написать udev правило, которое будет копировать на флеш-накопитель с заданным ID логи загрузки системы.

Udev правило, решающее задачу представлено в листинге 1

Листинг 1

```
ACTION=="add",
ATTRS{serial}=="408D5CBEC94AB471998546ED",
RUN+="/usr/bin/add_trigger.sh"
```

Запускаемый правилом сценарий bash представлен в листинге 2

Листинг 2

```
#!/bin/bash
sudo mount /dev/sdf1 /media/usb
sudo cat /var/log/boot.log > /media/usb/logs/boot.log
sudo cat /var/log/dmesg > /media/usb/logs/dmesg.log
sudo umount /media/usb
```

Результаты работы программы (содержимое флеш-накопителя просматривалось с ОС Windows во момент составления отчета):

KINGSTON (H:) > logs

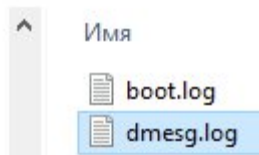


Рисунок 1 – просмотр файлов на флеш-накопителе

```
boot.log x dmesg.log x
1 |----- Mon Jul 05 22:59:56 MSK 2021 -----
2 | [<0x1b>[0;32m OK <0x1b>[0m Started <0x1b>[0;1;39mShow Plymouth Boot Screen<0x1b>[0m.
3 | [<0x1b>[0;32m OK <0x1b>[0m Started <0x1b>[0;1;39mForward Password R...s to Plymouth Directory Wat
4 | [<0x1b>[0;32m OK <0x1b>[0m Reached target <0x1b>[0;1;39mLocal Encrypted Volumes<0x1b>[0m.
5 | [<0x1b>[0;32m OK <0x1b>[0m Started <0x1b>[0;1;39mNetwork Time Synchronization<0x1b>[0m.
6 | [<0x1b>[0;32m OK <0x1b>[0m Reached target <0x1b>[0;1;39mSystem Time Set<0x1b>[0m.
7 | [<0x1b>[0;32m OK <0x1b>[0m Reached target <0x1b>[0;1;39mSystem Time Synchronized<0x1b>[0m.
8 | [<0x1b>[0;32m OK <0x1b>[0m Listening on <0x1b>[0;1;39mLoad/Save RF ...itch Status /dev/rfkill Wat
9 | [<0x1b>[0;32m OK <0x1b>[0m Started <0x1b>[0;1;39mNetwork Name Resolution<0x1b>[0m.
10| [<0x1b>[0;32m OK <0x1b>[0m Reached target <0x1b>[0;1;39mHost and Network Name Lookups<0x1b>[0m.
```

Рисунок 2 – фрагмент содержимого файла boot.log

1.2 Драйверы в Linux

Драйвер — программное обеспечение, с помощью которого другое программное обеспечение (операционная система) получает доступ к аппаратному обеспечению некоторого устройства.

Основные типы драйверов: символьные драйверы, блочные драйверы, сетевые драйверы.

Символьный драйвер обрабатывает поток байт.

Задание: написать драйвер, который инкрементирует внутреннюю переменную при каждом считывании.

Фрагмент исходного кода драйвера (на языке C), решающего поставленную задачу представлен в листинге 3

Листинг 3

```
#include <linux/kernel.h>
#include <linux/module.h>

// <...>подключение других библиотек

#define SUCCESS 0
#define DEVICE_NAME "test"
```

```

static int device_open( struct inode *, struct file * );
static int device_release( struct inode *, struct file * );
static ssize_t device_read( struct file *, char *, size_t,
loff_t * );

static int major_number;
static int is_device_open = 0;
static int counter = 0;

static struct file_operations fops =
{
    .read = device_read,
    .open = device_open,
    .release = device_release
};

/*
Реализация функций
static int __init test_init( void )
static void __exit test_exit( void )
static int device_open( struct inode *inode, struct file
*file )
static int device_release( struct inode *inode, struct file
*file )
*/

static ssize_t device_read(struct file *filp, char *buffer,
size_t length,
                        loff_t *offset)
{
    sprintf(buffer, "%d\n", counter++);
    printk( "Read driver tick.\n" );
    put_user( counter , buffer);
    return counter;
}

```

Для проверки работоспособности драйвера была написана программа, производящая циклическое чтение с заданного устройства. Ее исходный код представлен на листинге 4.

Листинг 4

```
#include <unistd.h>
#include <stdio.h>

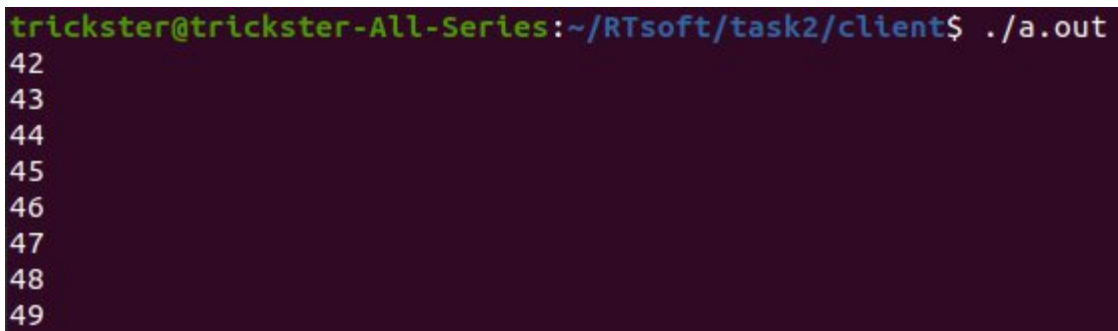
#define O_RDONLY          00

int main(){
    int fd;
    fd = open("/dev/test", O_RDONLY);
    while(1) {
        sleep(1);
        unsigned int t = 123;
        unsigned int t2 = 456;

        t2 = read(fd, &t, sizeof(t));

        printf("%d\n", t2);
    }
}
```

Результаты работы тестовой программы представлены на рисунке 3.



```
trickster@trickster-All-Series:~/RTsoft/task2/client$ ./a.out
42
43
44
45
46
47
48
49
```

Рисунок 3 - результаты работы тестовой программы

1.3 Контейнеризация

Контейнеризация (виртуализация на уровне операционной системы, контейнерная виртуализация, зонная виртуализация) — метод виртуализации, при котором ядро операционной системы поддерживает несколько изолированных экземпляров пространства пользователя вместо одного.

При этом при контейнеризации отсутствуют дополнительные ресурсные накладные расходы на эмуляцию виртуального оборудования и запуск полноценного экземпляра операционной системы, характерные при аппаратной виртуализации.

Docker — программное обеспечение для автоматизации развёртывания и управления приложениями в средах с поддержкой контейнеризации, контейнеризатор приложений.

Задание: создать docker-контейнер, содержащий браузер Mozilla firefox и запускающий его в графическом режиме.

Dockerfile, описывающий требуемый контейнер приведен в листинге 5.

Листинг 5

```
FROM
centos:8

RUN yum install firefox -
y

RUN yum install -y libcanberra-
gtk2

CMD firefox
```

Dockerfile устанавливает в контейнер минимальную версию Linux дистрибутива Cent OS, затем устанавливает браузер, графические библиотеки, запускает браузер.

Перед запуском контейнера необходимо предоставить docker права доступа к оконной системе X Window System

```
$ sudo xhost +local:root
```

При запуске контейнера необходимо указать параметры сети и графического интерфейса

```
$ sudo docker run -it --env="DISPLAY" --net=host  
myfirefox
```

1.4 Компьютерное зрение

Компьютерное зрение (иначе техническое зрение) — теория и технология создания машин, которые могут производить обнаружение, отслеживание и классификацию объектов.

OpenCV — библиотека алгоритмов компьютерного зрения, обработки изображений и численных алгоритмов общего назначения с открытым кодом.

Оператор Кэнни (детектор границ Кэнни, алгоритм Кэнни) в дисциплине компьютерного зрения — оператор обнаружения границ изображения.

Задача: написать программу, производящую поиск и выделение границ контрастных объектов на видео.

Исходный код требуемой программы (на языке python) представлен в листинге 5.

Листинг 5

```
import cv2 as cv  
import numpy as np  
import time  
  
print("hello from CV app")  
  
cap = cv.VideoCapture('test.mp4')  
while cap.isOpened():  
    print("CV-loop")  
    ret, frame = cap.read()  
  
    # resize =====  
  
    img = frame  
    scale_percent = 60 # percent of original size  
    width = int(img.shape[1] * scale_percent / 100)
```

```

height = int(img.shape[0] * scale_percent / 100)
dim = (width, height)

resized = cv.resize(img, dim, interpolation =
cv.INTER_AREA)

frame = resized

# main part =====

# if frame is read correctly ret is True
if not ret:
    print("Can't receive frame (stream end?).
Exiting ...")
    break

gray = cv.cvtColor(frame, cv.COLOR_BGR2GRAY)

# countours
edged = cv.Canny(gray, 30, 200)
contours, hierarchy = cv.findContours(edged, \
cv.RETR_EXTERNAL, cv.CHAIN_APPROX_NONE)
cv.drawContours(frame, contours[0:7], -1, (0, 255,
0), 1)

# rectangles
for cnt in contours[0:3]:
    x,y,w,h = cv.boundingRect(cnt)

cv.rectangle(frame,(x,y),(x+w,y+h),(0,0,255),2)
cv.imshow('Contours', frame)

# =====

# makes video slower
time.sleep(0.1)
if cv.waitKey(1) == ord('q'):
    break

cap.release()
cv.destroyAllWindows()

print("buy from CV app")

```

Программа обрабатывает каждый кадр видео в следующем порядке:

1. Масштабирование размеров изображения
2. Преобразование изображения к черно-белому
3. Получение граней предметов с помощью алгоритма Кэнни
4. Получение замкнутых контуров на основе данных о гранях
5. Отрисовка контуров на исходном изображении

Результат работы программы представлен на рисунке 4

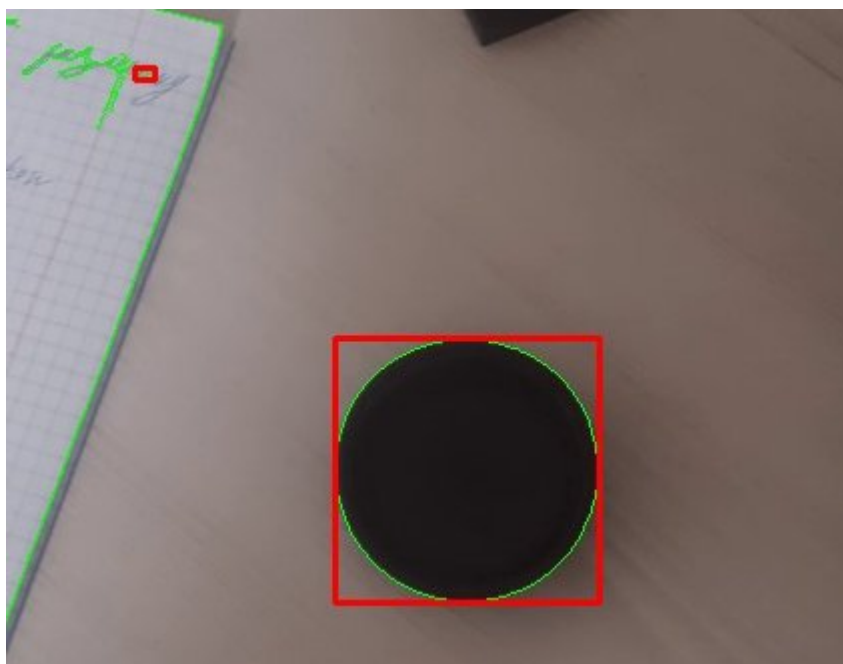


Рисунок 4 – поиск контуров контрастного объекта

1.5 Сетевой протокол MQTT

MQTT — упрощённый сетевой протокол, работающий поверх TCP/IP, ориентированный на обмен сообщениями между устройствами по принципу издатель-подписчик.

Задание: усовершенствовать программу распознавания контрастных объектов — отслеживать их траекторию (со сглаживанием) и публиковать информацию о координатах объекта по MQTT.

Функции, обеспечивающие вычисление и сглаживание траектории приведены на листинге 6.

Листинг 6

```
def running_mean(x, N):
    # x == an array of data
    # N == number of samples per average
    cumsum = np.cumsum(np.insert(x, 0, 0))
    return (cumsum[N:] - cumsum[:-N]) / float(N)

def pts_avg(pts, N):
    # pts == a deque of points
    # N == number of samples per average
    pts_list = list(pts)
    list_x = []
    list_y = []

    for elem in pts_list:
        list_x = np.append(list_x, list(elem)[0])
        list_y = np.append(list_y, list(elem)[1])

    x_avg = running_mean(list_x, min([len(list_x), N]))
    y_avg = running_mean(list_y, min([len(list_y), N]))

    pts2 = deque(maxlen=124)

    for i in range(0, len(x_avg)):
        pts2.append((int(x_avg[i]), int(y_avg[i])))

    return pts2
```

Код основной программы, производящей вызов данных функций и отправку MQTT-сообщений приведен на листинге 7.

Листинг 7

```
if __name__ == "__main__":

    print("hello from CV app")
    pts = deque(maxlen=124)
    cap = cv.VideoCapture('test2.mp4')
    i = 0

    client = mqtt.Client("cvSubscriber1")
    status = client.connect("localhost")
    if status == 0:
        print("MQQT connected")
    else:
        print("MQQT is down")

    while cap.isOpened():
        # print("CV-loop")
        ret, frame = cap.read()

        # if frame is read correctly ret is True
        if not ret:
            print("Can't receive frame (stream end?).
Exiting ...")
            break

        # main part =====

        frame = resize_img(frame, 60)
        contours2 = get_contours(frame, 7)

        for cnt in contours2:
            x,y,w,h = cv.boundingRect(cnt)
            if frame.shape[1]/w < 30 and
frame.shape[0]/h < 30:

                cv.rectangle(frame,(x,y),(x+w,y+h),
(0,0,200),2)
                center=(int(x+w/2),int(y+h/2))
                pts.appendleft(center)
```

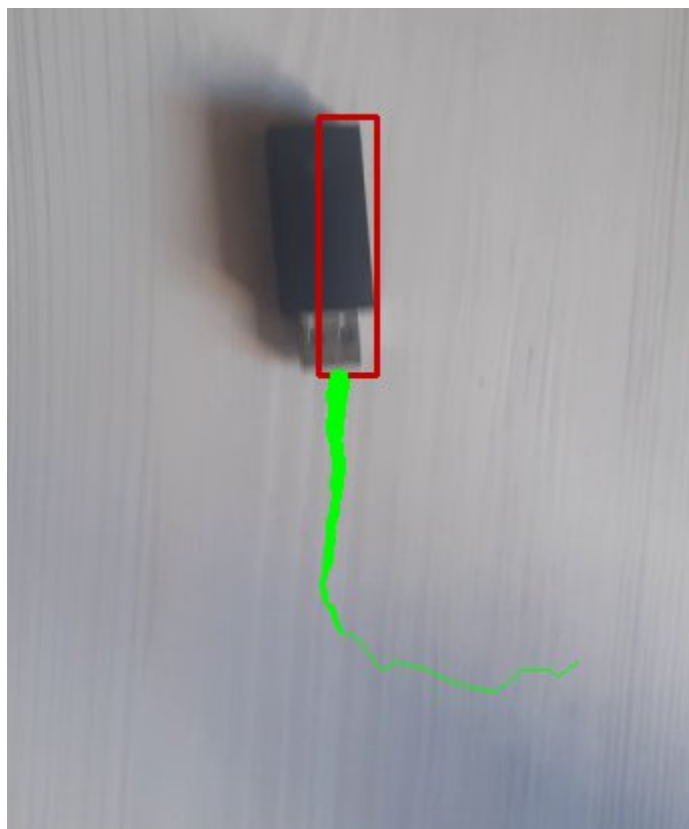



Рисунок 7 – визуализация траектории объекта

1.6 Apache Kafka

Брокер сообщений — архитектурный паттерн в распределённых системах; приложение, которое преобразует сообщение по одному протоколу от приложения-источника в сообщение протокола приложения-приёмника, тем самым выступая между ними посредником. Кроме преобразования сообщений из одного формата в другой, в задачи брокера сообщений также входит:

- проверка сообщения на ошибки;
- маршрутизация конкретному приемнику(ам);
- разбиение сообщения на несколько маленьких, а затем агрегирование ответов приёмников и отправка результата источнику;
- сохранение сообщений в базе данных;
- вызов веб-сервисов;
- распространение сообщений подписчикам, если используются шаблоны типа издатель-подписчик.

Apache Kafka — распределённый программный брокер сообщений, проект с открытым исходным кодом, разрабатываемый в рамках фонда Apache.

Задание: установить и протестировать работу Apache Kafka.

После установки Apache Kafka, необходимо запустить программы, приведенные на листингах 8 и 9 (“подписчик” и “издатель”).

Листинг 8

```
#!/bin/bash

systemctl start kafka ||
/home/kafka/kafka/bin/kafka-topics.sh --create --
zookeeper localhost:2181 --replication-factor 1 --
partitions 1 --topic TutorialTopic ||
Reset

echo 'listening Kafka messages:'
/home/kafka/kafka/bin/kafka-console-consumer.sh --
bootstrap-server localhost:9092 --topic TutorialTopic
```

Листинг 9

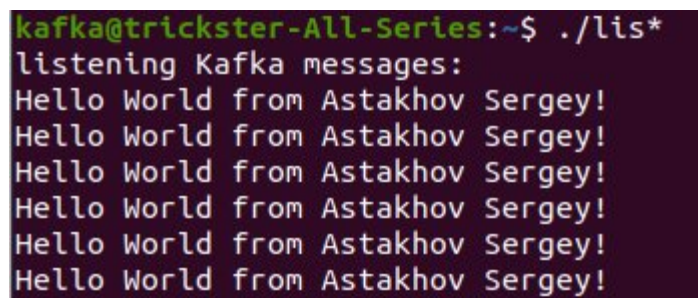
```
#!/bin/bash

echo 'begin sending 1 message per second'
while true
do

    echo "Hello World from Astakhov Sergey!" |
    /home/kafka/kafka/bin/kafka-console-producer.sh --
    broker-list localhost:9092 --topic TutorialTopic >
    /dev/null

    sleep 3
done
```

“Издатель” будет раз в секунду отправлять сообщение брокеру, а “подписчик” будет это сообщение считывать. Результаты работы программы отражены на рисунке 8.

A screenshot of a terminal window with a dark background. The prompt is 'kafka@trickster-All-Series:~\$'. The user has entered './lis*', and the output shows 'listening Kafka messages:' followed by six lines of 'Hello World from Astakhov Sergey!'.

```
kafka@trickster-All-Series:~$ ./lis*
listening Kafka messages:
Hello World from Astakhov Sergey!
Hello World from Astakhov Sergey!
Hello World from Astakhov Sergey!
Hello World from Astakhov Sergey!
Hello World from Astakhov Sergey!
Hello World from Astakhov Sergey!
```

Рисунок 8 – сообщения Kafka

1.7 TSDB - Базы данных временных рядов

Базы данных временных рядов (TSDB) представляет собой программное обеспечение системы, которая оптимизирована для хранения и подачи временных рядов через соответствующие пары времени (ы) и значения (ов).

Хотя можно хранить данные временных рядов во многих различных типах баз данных, конструкция этих систем со временем в качестве ключевого индекса заметно отличается от реляционных баз данных, которые уменьшают дискретные отношения с помощью ссылочных моделей.

Уникальные свойства наборов данных временных рядов означают, что базы данных временных рядов могут значительно улучшить пространство для хранения и производительность по сравнению с базами данных общего назначения.

Системы баз данных временных рядов построены так, чтобы быстро и эффективно принимать данные. Реляционные базы данных тоже имеют большую скорость загрузки данных (от 20 000 до 100 000 строк в секунду). Тем не менее, приём не постоянен во времени. У реляционных баз данных есть один ключевой аспект, который делает их медленными при росте данных — индексы.

Задание: усовершенствовать программу, распознающую контрастный объект так, чтобы данные из сообщений Kafka записывались в `influxDB` и визуализировались в `Grafana`.

Исходный код модуля, производящего запись данных в `influxDB` приведен в листинге 10

Листинг 10

```
from kafka import KafkaConsumer
from influxdb import InfluxDBClient
import time
import json

print("kafka -> influx bridge init")

consumer = KafkaConsumer(
    'cvcoords',
    bootstrap_servers=['localhost:9092'],
    enable_auto_commit=True)

client = InfluxDBClient(host='localhost', port=8086)
client.create_database('cvdata')
client.switch_database('cvdata')
fl = client.query('Delete FROM cv_measurement WHERE
time > 0')

print("kafka consumer loop init")

while True:
    time.sleep(0.01)
    for message in consumer:
        message = message.value.decode("utf-8")
        msg_json = json.loads(message)
        json_body = [
            {
                "measurement": "cv_measurement",
                "fields":{
                    "x": msg_json['x corrected'],
                    "y": msg_json['y corrected'],
                    "xbad": msg_json['x'],
                    "ybad": msg_json['y'],
                }
            }
        ]

        print(json_body)
        flag = client.write_points(json_body)
        print(flag)
```

Рисунки 9 и 10 демонстрируют содержимое influxDB и его визуализацию в Grafana.

```
> select * from "cv_measurement"
name: cv_measurement
time                x    xbad y    ybad
----                -    -   -    -
1625776747685059711 216 216 178 178
1625776747855720341 217 211 203 186
1625776747945913984 217 217 203 202
1625776748062703457 217 217 203 202
1625776748182713020 210 196 212 199
1625776748456529476 210 196 212 199
1625776748488958449 210 196 212 199
1625776748538064453 206 184 213 222
1625776748656461112 203 179 217 243
```

Рисунок 9 – координаты объекта в influxDB



Рисунок 10 - график координат $x(t)$ и $y(t)$ с учетом коррекции по скользящему среднему

1.8 REST и http-запросы

Задание: усовершенствовать программу поиска контрастных объектов так, чтобы можно было останавливать/запускать видео с помощью http-запроса.

Программа будет состоять из трех модулей:

- генератор http-запросов
- http-сервер
- распознаватель контрастных объектов

Генератор http-запросов отправляет http-запрос на сервер, если пользователь ввел в консоль “pause” или “play”. Его исходный код представлен на листинге 11.

Листинг 11

```
import requests
import json

while True:
    print("\nEnter remote command (play/pause):")
    cmd = input()
    if cmd == "play" or cmd == "pause":
        msg = json.dumps({"mode": cmd})
        x = requests.post('http://localhost:8000',
data = msg)

        if x.status_code == 200:
            print("server replied 200")
        else:
            print("something is wrong")
```

Http-сервер хранит последний пришедший POST-запрос и выдает его содержание по GET-запросу. Его исходный код представлен на листинге 12.

Листинг 12

```
from http.server import HTTPServer,
BaseHTTPRequestHandler
from io import BytesIO
import json

class
SimpleHTTPRequestHandler(BaseHTTPRequestHandler):

    mode = 'play'

    def do_GET(self):
        self.send_response(200)
        self.end_headers()
        msg_json = json.dumps({"mode":
SimpleHTTPRequestHandler.mode})
        msg_bytes = bytes(msg_json, 'utf-8')
        self.wfile.write(msg_bytes)

    def do_POST(self):
        content_length = int(self.headers['Content-
Length'])
        body = self.rfile.read(content_length)
        self.send_response(200)
        self.end_headers()
        response = BytesIO()
        response.write(b'Post requests received')

        cmd = json.loads(body.decode('utf-8'))

        if cmd["mode"] == 'play' or cmd["mode"] ==
'pause':
            SimpleHTTPRequestHandler.mode =
cmd["mode"]

            print("got POST request")

httpd = HTTPServer(('localhost', 8000),
SimpleHTTPRequestHandler)

httpd.serve_forever()
```


В распознаватель контрастных объектов добавляется цикл, который выполняется пока сервер отвечает, что программа должна держать видео на паузе. Его исходный код приведе на листинге 13.

Листинг 13

```
# внешний цикл

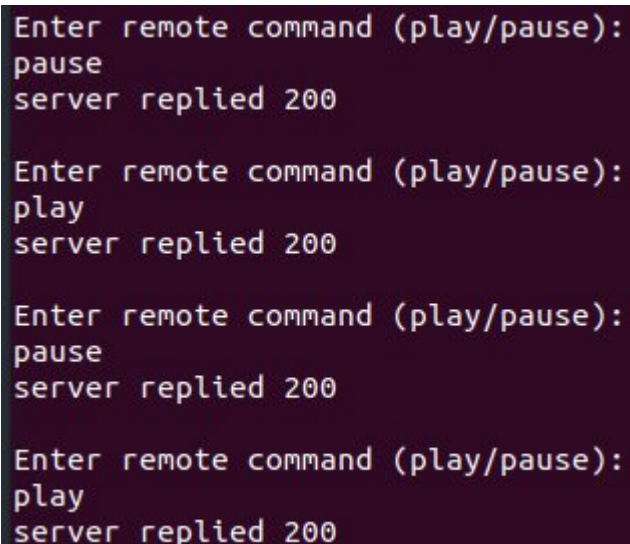
fl_log = True

    while get_http_cmd() == 'pause':
        time.sleep(0.5)
        if fl_log:
            print("paused")
            fl_log = False

    if not fl_log:
        print("played")

# внешний цикл
```

Результаты работы программы можно отследить по отладочным сообщениям модулей, приведенным на рисунках 11-13.



```
Enter remote command (play/pause):
pause
server replied 200

Enter remote command (play/pause):
play
server replied 200

Enter remote command (play/pause):
pause
server replied 200

Enter remote command (play/pause):
play
server replied 200
```

Рисунок 11 - Интерактивная консоль управляющей программы (генератора http-запросов)

```
trickster@trickster-All-Series:~/RTsoft/task8/src$ python3 server.py
127.0.0.1 - - [09/Jul/2021 21:16:11] "POST / HTTP/1.1" 200 -
got POST request
127.0.0.1 - - [09/Jul/2021 21:16:22] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [09/Jul/2021 21:16:22] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [09/Jul/2021 21:16:22] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [09/Jul/2021 21:16:23] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [09/Jul/2021 21:16:23] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [09/Jul/2021 21:16:23] "GET / HTTP/1.1" 200 -
127.0.0.1 - - [09/Jul/2021 21:16:23] "GET / HTTP/1.1" 200 -
```

Рисунок 12 – сообщения http-сервера

```
trickster@trickster-All-Series:~/RTsoft/task8/src$ python3 main.py
hello from CV app
MQQT connected
paused
played
paused
played
good bye from CV app
trickster@trickster-All-Series:~/RTsoft/task8/src$
```

Рисунок 13 – сообщения основной программы

2 Индивидуальное задание

Задание: разработать конвертер сообщений Kafka в формат OPC UA. Исходные данные брать с эмулируемых или реальных датчиков. Визуализировать передаваемые значения.

Все модули реализованы на языке python.

Основная программа генерирует случайные данные и отправляет их в формате json внутри сообщения Kafka.

Сообщения Kafka принимаются вторым модулем и сохраняются на реализуемом модулем орс-сервере. Код модуля представлен в листинге 14. Содержимое орс-сервера может быть просмотрено с помощью специальной утилиты, пример приведен на рисунке 14.

Листинг 14

```
from kafka import KafkaConsumer
import json
import time
from opcua import Server
from opcua.ua import VariantType

URL = "opc.tcp://0.0.0.0:4840"

server = Server()
server.set_endpoint(URL)

objects = server.get_objects_node()
ns = server.register_namespace("My metrics")
accelerometer = objects.add_object(ns, "accelerometer")

x_metric = accelerometer.add_variable(ns, "x", 0.0,
varianttype = VariantType.Double)
y_metric = accelerometer.add_variable(ns, "y", 0.0,
varianttype = VariantType.Double)
z_metric = accelerometer.add_variable(ns, "z", 0.0,
varianttype = VariantType.Double)
```

```

termometer = objects.add_object(ns, "termometer")
t_metric = termometer.add_variable(ns, "temperature", 0.0,
varianttype = VariantType.Double)

server.start()

consumer = KafkaConsumer(
    bootstrap_servers=['localhost:9092'],
    enable_auto_commit=True)

consumer.subscribe(['coords','temperature'])

print("Converter loop init")

while True:
    for message in consumer:

        message_val = message.value.decode("utf-8")
        msg_json = json.loads(message_val)
        if message.topic == 'coords':
            x_metric.set_value(msg_json["x"], varianttype =
VariantType.Double)
            y_metric.set_value(msg_json["y"], varianttype =
VariantType.Double)
            z_metric.set_value(msg_json["z"], varianttype =
VariantType.Double)

            if message.topic == 'temperature':
                t_metric.set_value(msg_json["temperature"],
varianttype = VariantType.Double)

        time.sleep(0.05)

```

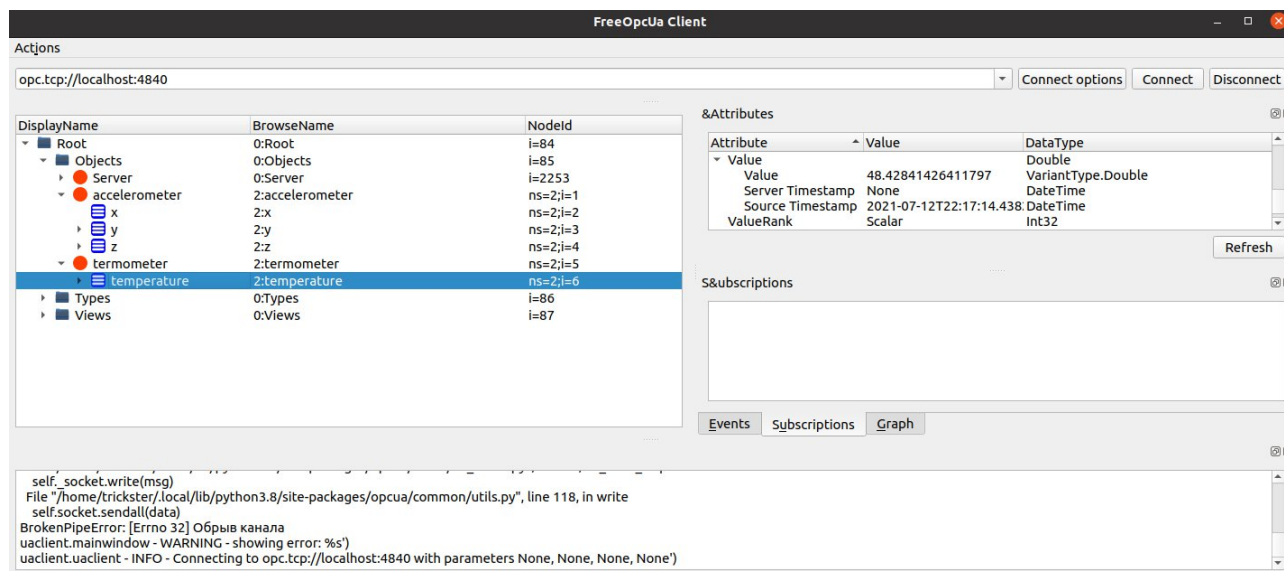


Рисунок 14 - узлы орс-сервера в окне программы орс-client

Третий модуль (листинг 15) с заданной периодичностью запрашивает данные с орс-сервера и записывает их в influxDB, далее они визуализируются с помощью Grafana (рисунок 15).

Листинг 15

```
import time
from opcua import Client
from influxdb import InfluxDBClient

db_client = InfluxDBClient(host='localhost',
port=8086)
db_client.create_database('opcdata')
db_client.switch_database('opcdata')
fl = db_client.query('Delete FROM accel WHERE time >
0')
fl = db_client.query('Delete FROM temperature WHERE
time > 0')

URL = "opc.tcp://localhost:4840"

if __name__ == "__main__":
    client = Client(URL)
    client.connect()

    xNode = client.get_node("ns=2;i=2")
    yNode = client.get_node("ns=2;i=3")
```

```

zNode = client.get_node("ns=2;i=4")
tNode = client.get_node("ns=2;i=6")
print("Client loop init")

while True:
    # print("Client listen 1 sec tick")

    x = xNode.get_value()
    y = yNode.get_value()
    z = zNode.get_value()
    t = tNode.get_value()

    json_body = [{"measurement": "accel",
                  "fields":{"x": x,"y": y,"z": z}}]

    flag = db_client.write_points(json_body)

    json_body = [{"measurement": "temperature",
                  "fields":{"t": t}}]

    flag = db_client.write_points(json_body)
    time.sleep(0.05)

```



Рисунок 15 - показания акселерометра в Grafana

ЗАКЛЮЧЕНИЕ

В течении летней практики были освоены навыки:

- работы с ОС Linux и написания драйверов для нее
- контейнеризации при помощи Docker
- работы с библиотекой компьютерного зрения OpenCV
- работы с брокерами сообщений MQTT и Kafka
- работы с TSDB(influxDB) и Grafana

В рамках итогового проекта была написана программа, принимающая kafka-сообщения, конвертирующая их в формат OPC-UA и публикующая на OPC-сервере. Задача выполнена успешно.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. InfluxData Documentation [Электронный ресурс]. URL: <https://docs.influxdata.com/> (дата обращения: 08.07.2021)
2. Documentation | Grafana Labs [Электронный ресурс]. URL: <https://grafana.com/docs/> (дата обращения: 10.07.2021)
3. Установка Apache Kafka на Ubuntu [Электронный ресурс]. URL: <https://www.digitalocean.com/community/tutorials/how-to-install-apache-kafka-on-ubuntu-18-04-ru> (дата обращения: 12.07.2021)
4. Kafka-python documentation [Электронный ресурс]. URL: <https://kafka-python.readthedocs.io/en/master/> (дата обращения: 15.07.2021)
5. Python OPC-UA documentation [Электронный ресурс]. URL: <https://python-opcua.readthedocs.io/en/latest/> (дата обращения: 19.07.2021)