|  |  |
| --- | --- |
|  | **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  предприятия | АО «РТСофт» |

|  |  |  |  |
| --- | --- | --- | --- |
| Студент ИУ6-42Б |  |  | С.В. Астахов |
|  |  | (Подпись, дата) | (И.О. Фамилия) |
|  |  |  |  |
| Руководитель практики |  |  | Е.В. Смирнова |
|  |  | (Подпись, дата) | (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.

**ЗАДАНИЕ**

**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 « 05 » июля 2021 г.

|  |  |  |  |
| --- | --- | --- | --- |
| **Руководитель практики** |  |  | Е.В. Смирнова |
|  |  | (Подпись, дата) | (И.О. Фамилия) |
| **Студент** |  |  | С.В. Астахов |
|  |  | (Подпись, дата) | (И.О. Фамилия) |

Примечание: Задание оформляется в двух экземплярах.

СОДЕРЖАНИЕ

[ВВЕДЕНИЕ 4](#_Toc1)

[ОСНОВНАЯ ЧАСТЬ 5](#_Toc2)

[1 Ознакомительные задания 5](#_Toc3)

[1.1 Udev правила 5](#_Toc4)

[1.2 Драйверы в Linux 6](#_Toc5)

[1.3 Контейнеризация 9](#_Toc6)

[1.4 Компьютерное зрение 11](#_Toc7)

[1.5 Сетевой протокол MQTT 14](#_Toc8)

[1.6 Apache Kafka 18](#_Toc9)

[1.7 TSDB - Базы данных временных рядов 20](#_Toc10)

[1.8 REST и http-запросы 23](#_Toc11)

[2 Индивидуальное задание 27](#_Toc12)

[ЗАКЛЮЧЕНИЕ 31](#_Toc13)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 32](#_Toc14)

# ВВЕДЕНИЕ

Цель производственной практики: изучение механизмов работы с внешнеми 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

# Ознакомительные задания

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

# 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):

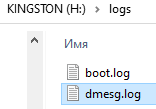


Рисунок 1 – просмотр файлов на флеш-накопителе

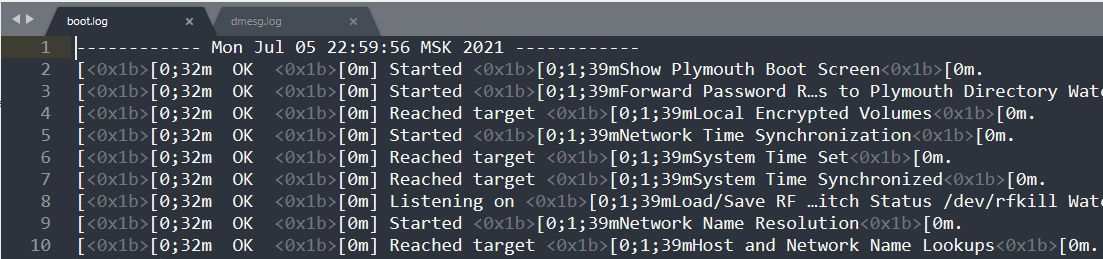


Рисунок 2 – фрагмент содержимого файла boot.log

# 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 С), решающего поставленную задачу предствален 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.

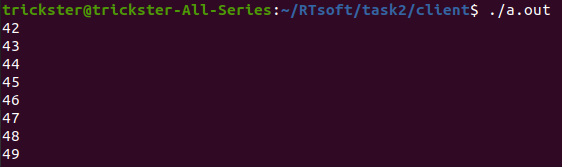


Рисунок 3 - результаты работы тестовой программы

# 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 диструбитва 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

# 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 изображение
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 изобржаении

Результат работы программы представлен на рисунке 4



Рисунок 4 – поиск контуров контрастного объекта

# 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)

avg\_pts = pts\_avg(pts, 10)

# draw\_track(pts, 1, (255,0, 0))

draw\_track(avg\_pts, 2, (0,255,0))

i += 1

if i % 10 == 0:

# print(avg\_pts[0])

x = list(pts[0])[0]

y = list(pts[0])[1]

x\_avg = list(avg\_pts[0])[0]

y\_avg = list(avg\_pts[0])[1]

msg = json.dumps({"x":x, "y":y, "x corrected":x\_avg, "y corrected": y\_avg})

# client.publish("cv/track", str(avg\_pts[0]))

client.publish("cv/track", msg)

# ===========================

# makes video slower

time.sleep(0.1)

if cv.waitKey(1) == ord('q'):

break

cap.release()

cv.destroyAllWindows()

print("good bye from CV app")

Результаты работы программы отражены на скриншотах 6 и 7

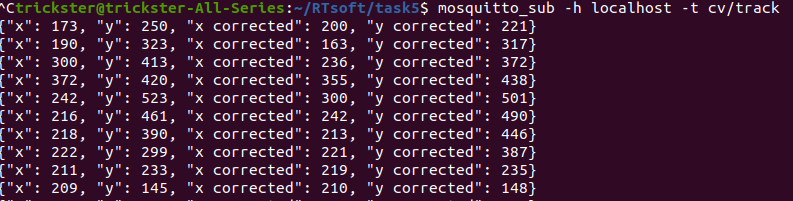


Рисунок 6 – MQTT-сообщения с координатами объекта

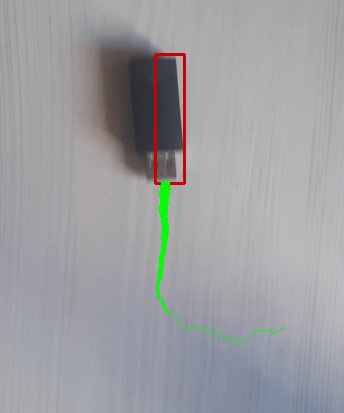


Рисунок 7 – визуализация траектории объекта

# 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.

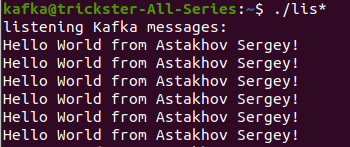


Рисунок 8 – сообщения Kafka

# 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.

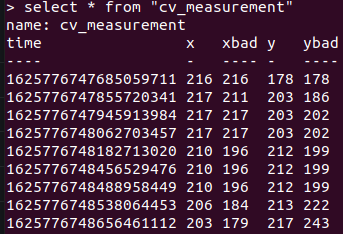


Рисунок 9 – координаты объекта в influxDB



Рисунок 10 - график координат х(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.

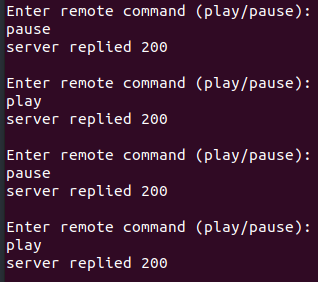


Рисунок 11 - Интерактивная консоль управляющей программы (генератора http-запросов)

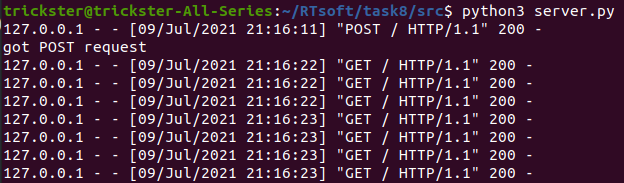


Рисунок 12 – сообщения http-сервера

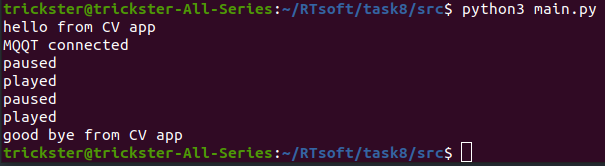


Рисунок 13 – сообщения основной программы

# 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 opc-сервере. Код модуля представлен в листинге 14. Содержимое opc-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)

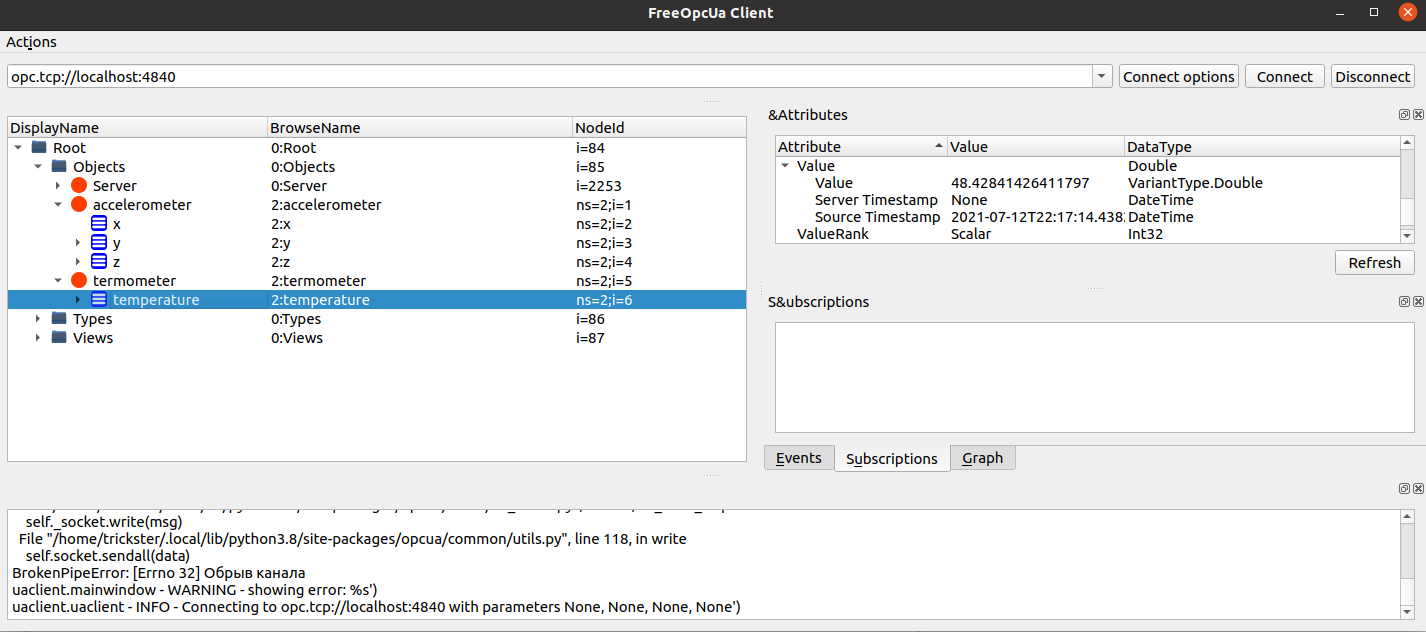


Рисунок 14 - узлы opc-сервера в окне программы opc-client

Третий модуль (листинг 15) с заданной переодичностью запрашивает данные с opc-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)