Санкт-Петербургский Политехнический Университет Петра Великого
Институт компьютерных наук и технологий
Высшая школа программной инженерии

КУРСОВАЯ РАБОТА

«Подсчёт машин»
«Бумажная клавиатура»

по дисциплине «Микропроцессорные системы»

Выполнили
студенты гр. 33534/5

Стойкоски Н. С.
Чинь В. Т.
Нгуен Х. И.
Нгуен Ч. К.
Хараки Ю.
Мулонде Ф.

Руководитель: Круглов С. К.

«___» _____ 2018 г.

Санкт-Петербург
2018

Содержание

Введение

Orange Pi i96 — это плата спецификации 96Boards IoT Edition с RDA8810PL SoC на базе ядра ARM Cortex-A5, WiFi и Bluetooth, USB и микро-USB и 40-контактный разъем GPIO. Плата Orange Pi i96 была впервые представлена на выставке Linaro Connect US 2016 в сентябре того же года и является второй платой 96Boards IoT Edition (IE) после BLE Carbon от SeeedStudio.

Orange Pi i96 представляет собой недорогую версию платы Orange Pi 2G-IoT без поддержки 2G / GSM, без интерфейса дисплея, без гнезда для наушников и меньшего размера печатной платы. Как и Orange Pi 2G-IOT, есть встроенный модуль RDA5991 WiFi/Bluetooth с внешней антенной, а также 256MB LPDDR2 RAM, 500MB NAND flash и слот microSD. Следуя спецификации IE (IoT Edition), есть порт USB 2.0 HOST и порт OTG с микро-USB. Также есть интерфейс CSI, поддерживающий до 5-мегапиксельную камеру.

С помощью платы и камеры, мы строим два проекта, которые помогают людям в жизни. Первый проект это «Подсчёт машин», совершенный модуль поставляется на входе/выходе парковки и на переходов от одного этажа на другого, наблюдаются входные и выходые машины, определяется количесто машин, количество свободных мест в парковке. Как результаат, можно будет доставить информацию водителям о свободних мест на каждом этаже при самого входа в парковку.

Другой проект называется «Бумажной клавиатурой». С помощью его, пользователь не должен всегда сидеть рядом с комьютером или телевизором, можеть даже лежать на кровати, и работать на бумажной клавиатуре, результат такой же как и работа на обычной клавиатуре.

Задание

1. Задача: Подсчёт машин

- Установить программу на Orange Pi i96, которая с помощью камеры подсчитает количество машин входных (выходных) в парковку, и поставить этот модуль на входе/выходе парковки. По результатам определить количество свободных мест и машин в парковке.
- Программа отправляет клиентскому приложению эту информацию о состояний парковки.

2. Задача: Бумажная клавиатура

- Установить программу на Orange Pi i96, которая с помощью камеры распознавает нажатую клавишу на листочке где напечатан образ клавиатуры.
- Программа эту информацию отправляет клиентскому приложению, которое от полученной информацию отправляет сигнал к операционной системе о нажатой клавиши.

Описание прибора

Нужные приборы:

- Orange Pi i96
- Камера для Orange Pi i96
- Микрокарта 16GB
- Адаптер 5В 2А, штекер 4.0 1.7

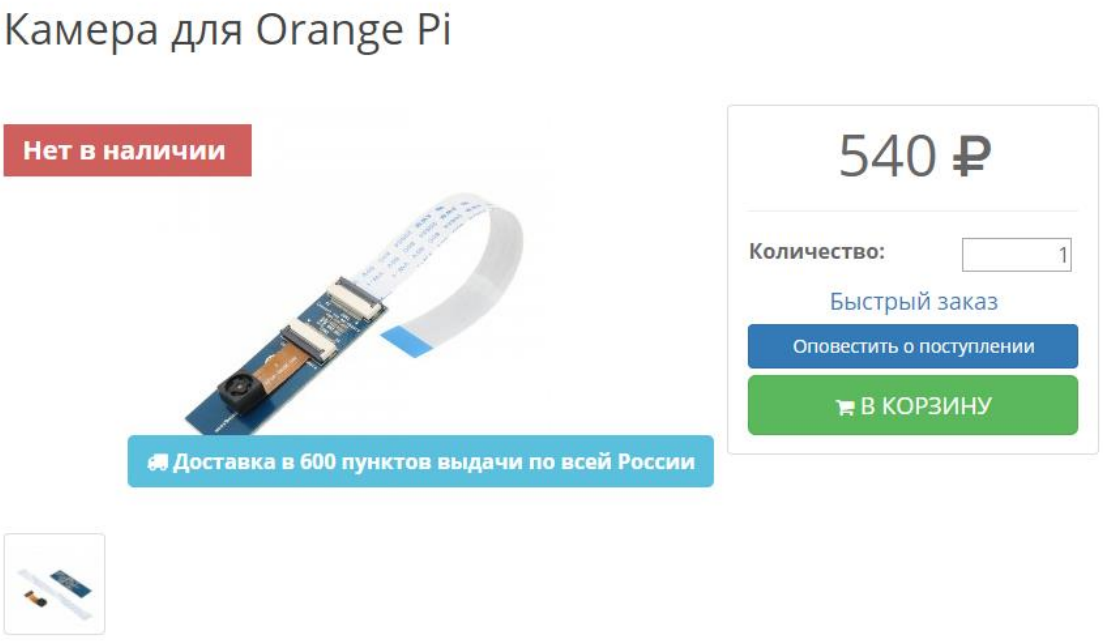
Orange Pi i96 - упрощенная версия контроллера Orange Pi 2g-IoT, без 2G, но все еще IoT. Модуль 2G забрал с собой и повышенное потребление, в итоге эта плата более адаптирована для IoT устройств, питающихся от батареи. Прямо на контроллере имеются контакты для подключения батареи, и он регулирует и заряд и разряд подключенного аккумулятора. Так же удалось еще больше уменьшить размер - теперь он составляет всего 60x30 мм, что делает его самым компактным контроллером серии Orange Pi.

Бюджет

1. Orange Pi i96.

https://roboshop.spb.ru/Orange-Pi-I96

2. Камера для Orange Pi i96.

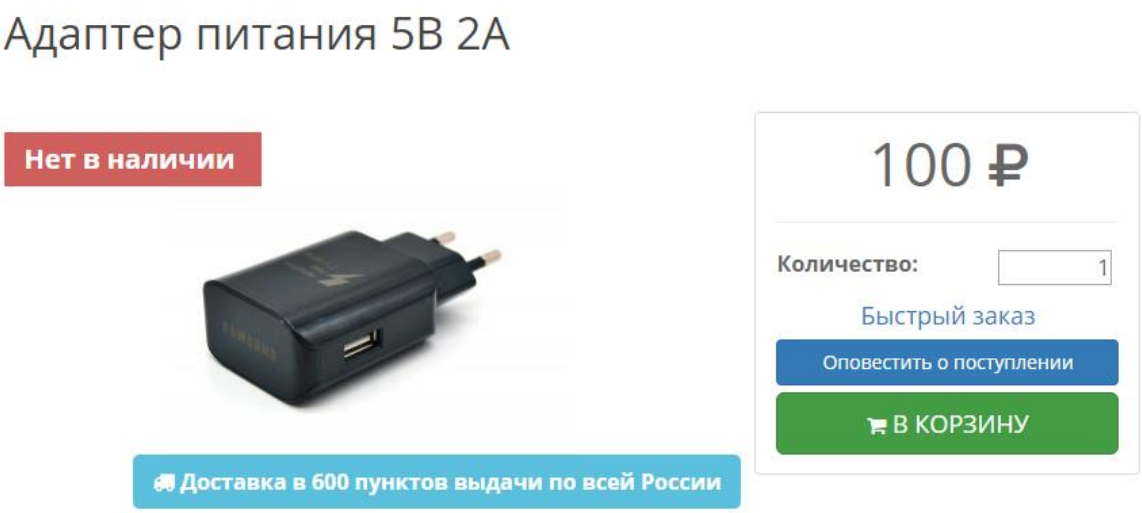


https://roboshop.spb.ru/orange-pi-camera

3. Микрокарта 16GB.

https://www.eldorado.ru/cat/detail/71358600/?utm_term=

4. Адаптер 5В 2А.



https://roboshop.spb.ru/usb-power-5v-2a-fast-charging

- Orange pi i96 1020 рублей
- Камера для Orange pi i96 540 рублей
- Адаптер 5В 2А 100 рублей
- Кабель (штекер 4.0 1.7) 120 рублей

Итого: 1800 рублей

Характеристики устройств

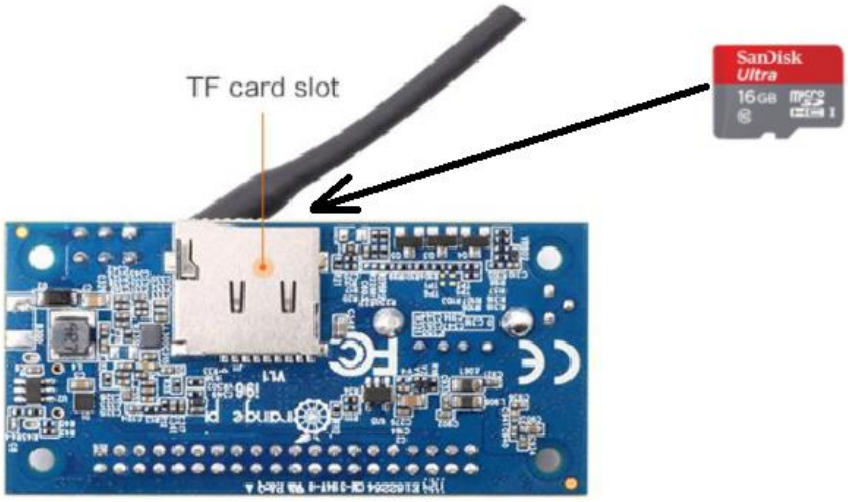
Характеристики Orange pi i96:

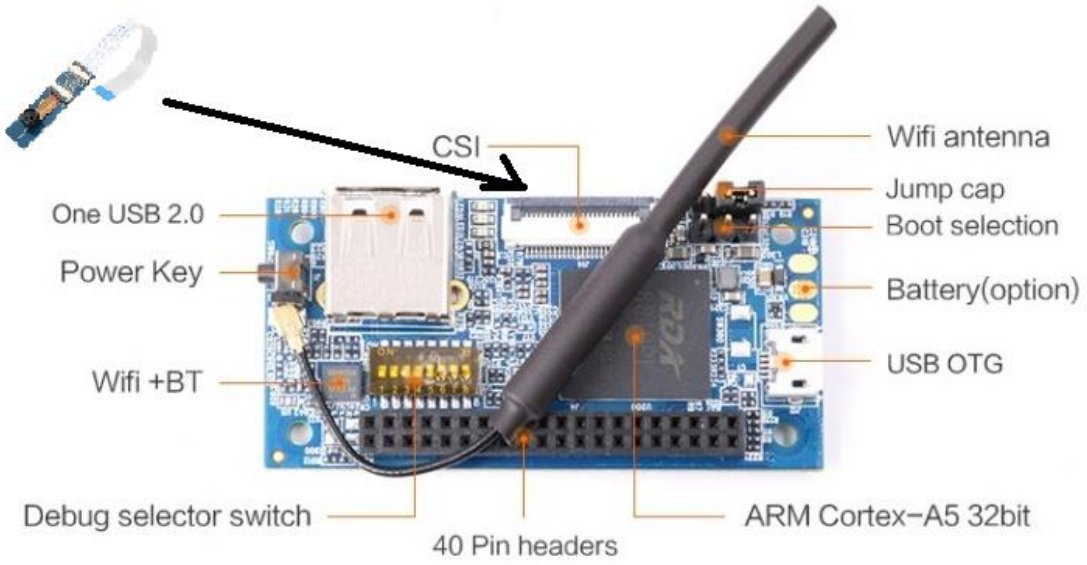
Процессор	1000 МГц Cortex-A5 32 бита
GPU	Vivante's GC860 Поддерживает OpenGLES1.1/2.0 Поддерживает OpenVG1.4 Поддерживает DirectFB Поддерживает GDI/DirecShow 30M Triangle/s, 250M Pixel/s
Оперативная память	LPDDR2 256 Мб (совместно с GPU)
Поддерживаемые карты памяти	Micro SD, есть версия со встроенной SLC NAND флэш-памятью 500 Мб
Wi-Fi и Bluetooth	Да, Wi-Fi 802.11 b/g/n и Bluetooth 2.1 на RDA5991
Видео-вход	разъём CSI для камеры. Поддерживает 8-bit YUV422 CMOS сенсорный интерфейс Поддерживает CCIR656 протокол для NTSC и PAL Поддерживает SM pixel сенсор Поддерживает разрешение 1080p@30fps
Источник питания	USB OTG
Порты USB	USB 2.0 x 1 штука, USB OTG x 1 штука
Другие разъёмы	40-пиновый разъём
GPIO (1x3)	UART, земля
Светодиоды	Индикатор питания
Кнопки	Включение
Поддерживаемые ОС	Android, Ubuntu, Debian, Raspbian, Linux
Размер	60 мм на 30 мм
Вес	30 грамм

Характеристики миниатюрного модуля камеры на 2Мп сенсоре GS2035, официальная камера для Orange Pi:

- Active pixel array 1616 x 1232
- ADC resolution 10 bit ADC
- Shutter type Electronic rolling shutter
- Max Frame rate 15fps@24Mhz, UXGA
- ~30fps@24Mhz, SVGA
- Power Supply AVDD28: 2.7~3.0V
- DVDD18: 1.7~1.9V
- IOVDD: 1.7~3.0V
- Power Consumption 180mW(Active)
- <100uA(Standby)
- SNR TBD
- Dark Current TBD
- Sensitivity TBD
- Operating temperature: -20~70℃
- Stable Image temperature 0~50℃

Схемы устройства





Описание приложения

1. Приложение: Подсчёт машин

Это приложение устанавливается на модули стоящий на входе/выходе парковки. С помощью камеры, оно считает количество машин, которые прошли дверь, из этого определяет количество свободных и занятых мест в парковке. И потом отправляет количество машин клиентскому приложению.

Клиентское приложение читает количество машин в парковке. По этим результатам можно будет доставить информацию водителям о свободних мест на каждом этаже при самого входа в парковку.

Программа написана на языке C++. Для подсчёта машин, используется библиотека OpenCV. Смотреть машину как прямоугольник, который имеет размер похожен на машину. И вместо двери нарисована линия. Приямоугольник имеет координат пересекает с координатом этой линии, читает что эта машина прошла дверь.

Для передачи количества машин, программа использует TCP/IP протокол. Отправляет результат на адрес IP/Port. И из этого адреса IP/Port, читает результат.

2. Приложение: Бумажная клавиатура

Это приложение устанавливается на модуль который с помощью камеры, распознавает нажатую клавишу на листочке где напечатан образ клавиатуры. Программа эту информацию в символьном виде отправляет клиентскому приложению.

Клиентское приложение от полученной информацию о нажатой кнопки симулирует настояшое нажатие кнопки с использованием функции из библиотеку Xlib.

Программа написана на языке C++. Для распознавание нажатой кнопки используется библиотека OpenCV. Распечатанную клавиатуру, программа видит как трапеция, по нахождение геометрических характеристик которой, получает информацию об области которой занимает каждая клавиша в кадре. При этом легко строится соотвествие область(ячейка)-клавиша в символьном виде. В каждой ячейке сравнивается состояние значения цвета пикселя с значениям полученных при этапе калибрации.

Для передачи информацию об нажатой клавишу, программа использует TCP/IP протокол. Отправляет результат на адрес IP/Port. И из этого адреса IP/Port, читает воспроизводит нажатие кнопки в операционной системы.

Установка

Установка Linux на Orange Pi i96

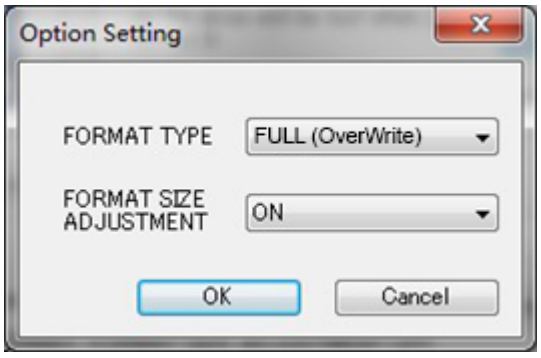
Шаг 1. Подготовка карты.

1.1 Вставьте карту памяти в свой персональный компьютер на Windows. Помните, что карта должна быть не менее 4 Гб, а скорость чтения не ниже 10-ого класса. Лучше всего подходят карты SanDisk.

1.2 Скачайте программу "SD Formatter" по этой ссылке http://orangepi.su/files/SDFormatterv4.zip

1.3 Разархивируйте архив с программой "SD Formatter".

1.4 Запустите SD Formatter, зайдите в "Options" и в открывшемся окне задайте "FORMAT TYPE" выбрать "FULL (OverWrite)", а в графе "FORMAT SIZE ADJUSTMENT" выбрать "ON". Далее нажмите "OK".



1.5 Нажмите "Format" и ждите полного форматирования карты.

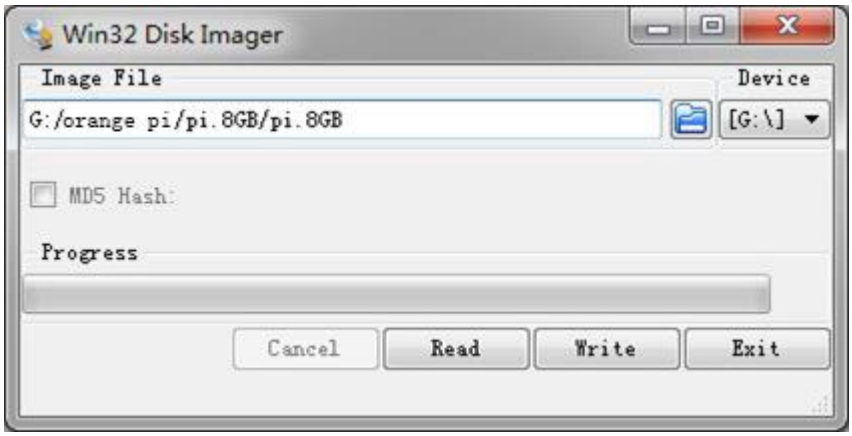
Шаг 2. Закачка ОС.

2.1 Скачайте дистрибутив операционной системы (Linux) для вашей модели Orange Pi по этому
адресу http://www.orangepi.org/downloadresources/

2.2 Разархивируйте полученный архив.

2.3 Скачайте "Win32 Diskimage" по этому
адресу http://orangepi.su/files/Win32DiskImager-0.9.5-install.rar

2.4 Разархивируйте "Win32 Diskimage" и запустите его. В открывшемся окне укажите путь к скаченной в пункте 2.1 операционной системе.



2.5 Нажмите "Write" и ждите когда завершится процесс записи.

Шаг 3: Запуск Orange Pi

Готово! Теперь вставьте SD-карту в микрокомпьютер Orange Pi, подключить к камере и включите питание. Помните, что первый запуск длитеся дольше обычного и может занять несколько минут!

Логин и пароль по умолчанию: root/orangepi.

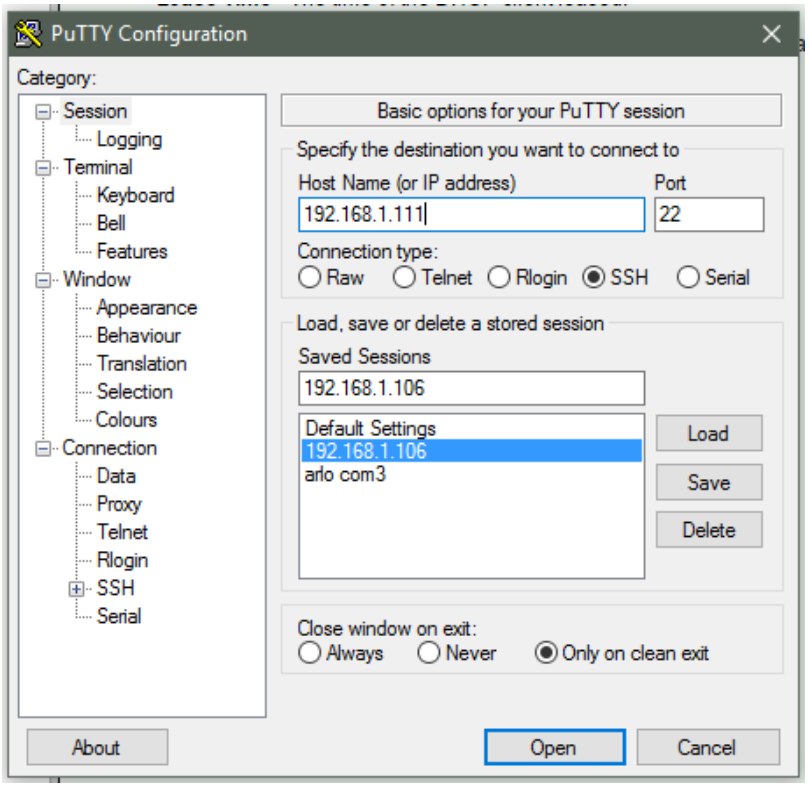
Установка PuTTY Configuration

Для запуска Orange Pi i96, посколько у нас нет монитора, мы используем PuTTY Configuration по SSH.

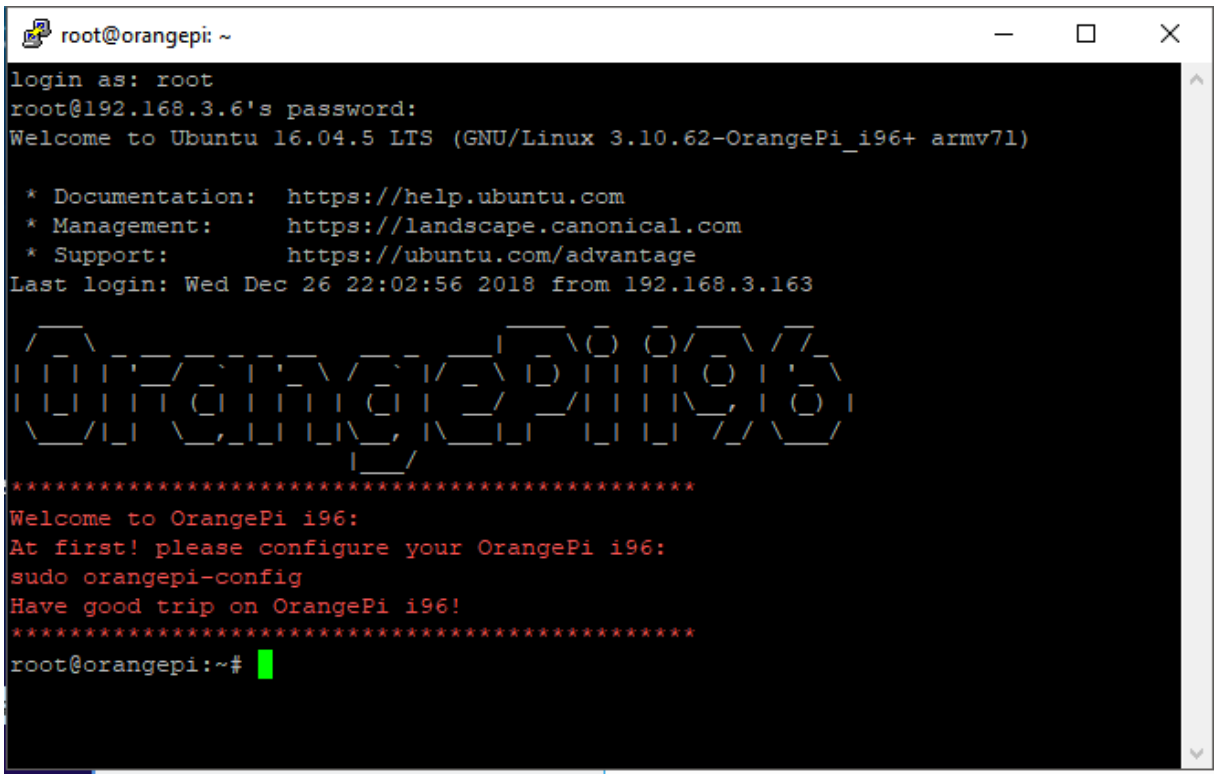
1/ Скачайте программу Putty по этой ссылке
https://www.chiark.greenend.org.uk/~sgtatham/putty/latest.html

2/ Находим IPV4 адрес нашего устройства в приложение сетевого рутера.

3/ Запускаем Putty, вводим в ней этот Ip адресс и нажимаем кнопку Open



Если все сделали правильно, в окне терминала появится запрос логина и пароля



Подключение к платы. Логин и пароль: root/ orangepi.

Установка OpenCV на Orange Pi i96

1/ Запускай команду для установки.

```
sudo apt-get install libopencv-dev
```

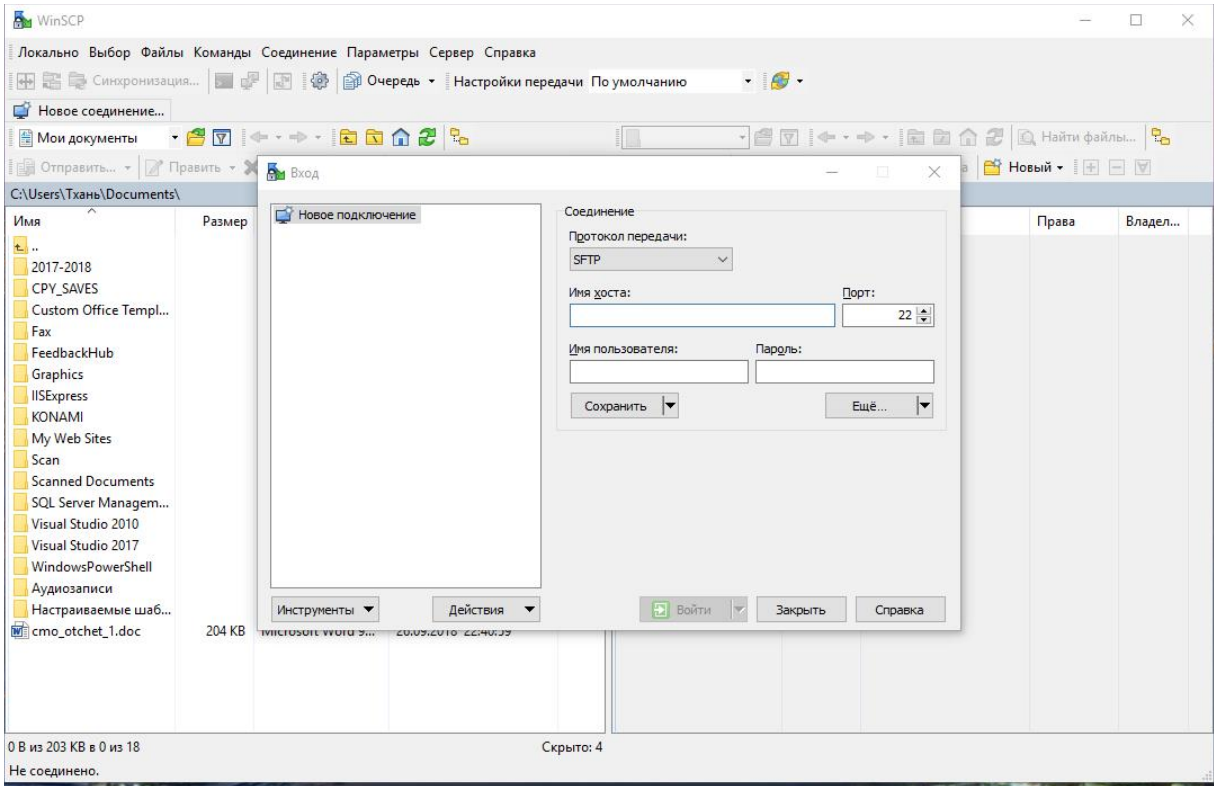
2/ Для проверки запускай команду.

```
pkg-config --modversion  opencv
```

Установка WinSCP для передачи файл между компьютером и ОС платой Orange Pi

1/ Скачайте программу "Win SCP" по этой ссылке https://winscp.net/eng/download.php

2/ Запустите WINSCP, вводите IP адреса платы, имя пользователя root, пароль orangepi.



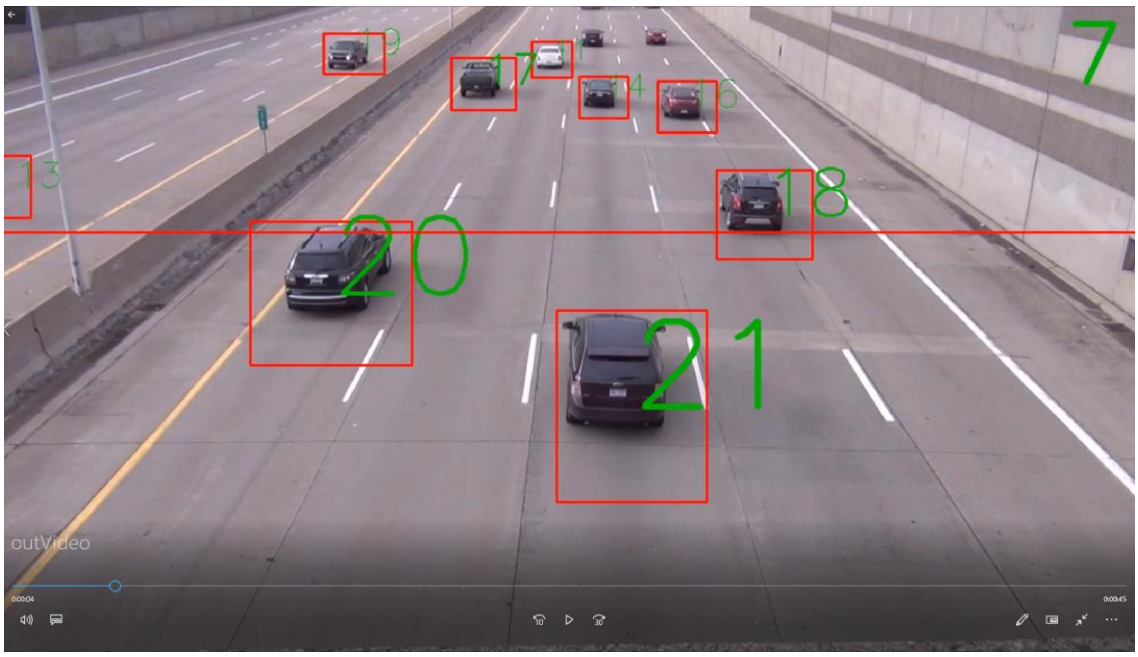
Сейчас можете передать файл между комьютером и платой. Мы используем для передачи проекта, который написан на Visual Studio 2017 в Orange Pi, и запускать его.

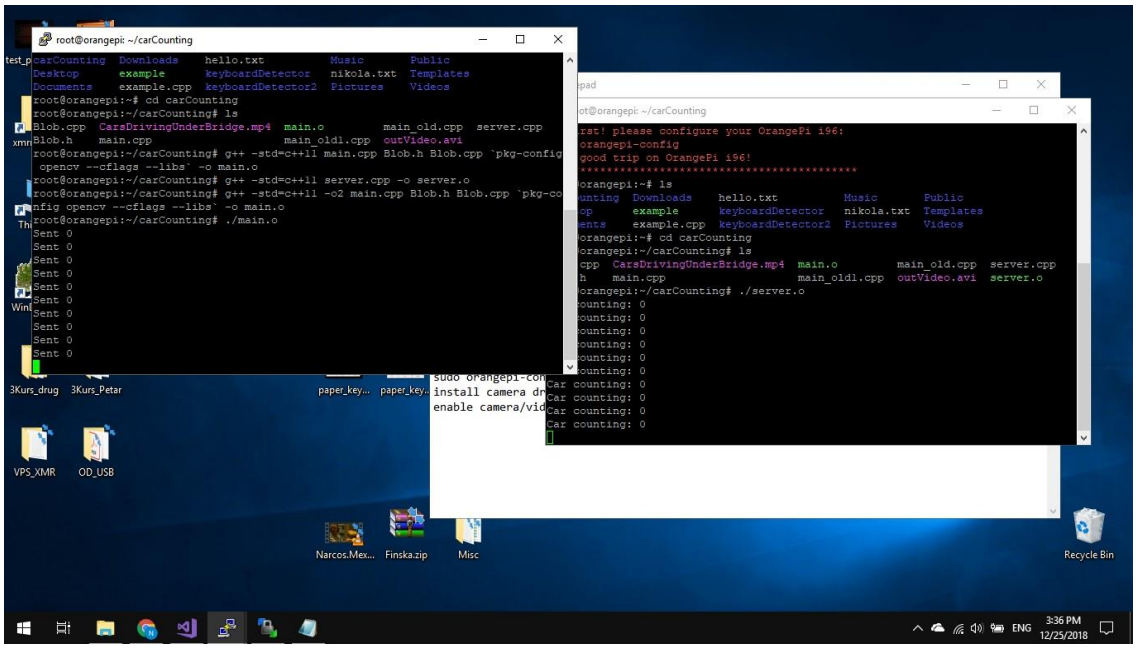
Полученные результаты

1. Приложение: Подсчёт машин

Команда запускает программу:

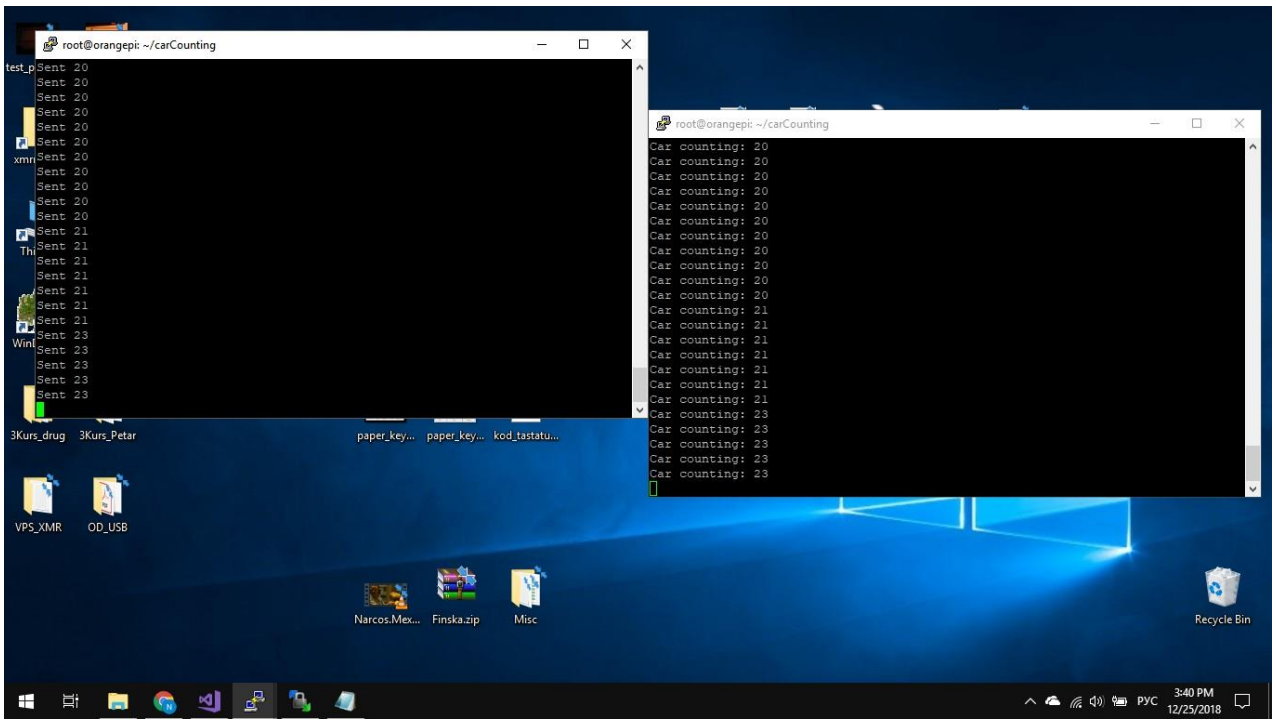
```
# g++ -std=c++11 main.cpp Blob.h Blob.cpp `pkg-config opencv --cflags --libs` -o
main.o


# ./main.o
```





Команда запускает сервер:

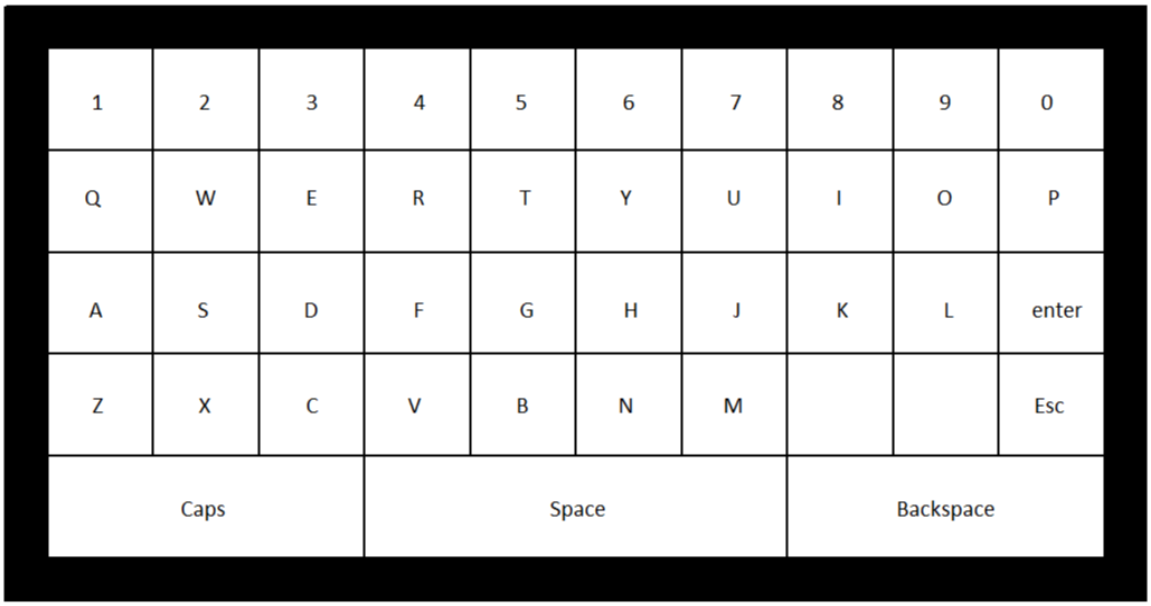
```
# g++ client.cpp -o client.o


# ./client.o
```



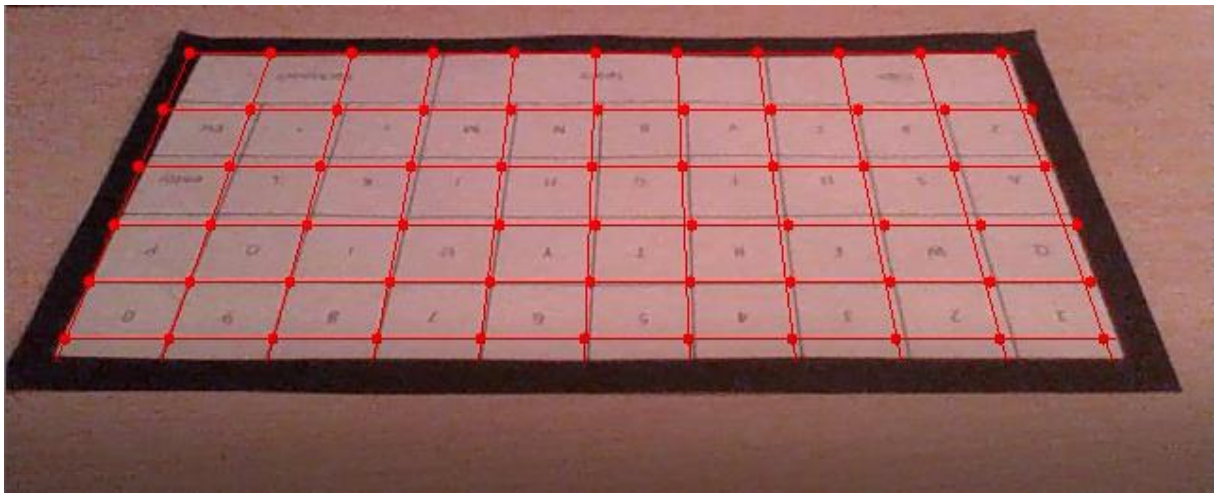
Для проверки работы программы, на входе мы поставляем видео, снимает на улице, и программа читает количество машин в видео и отправляет на другой программе результат, эта программа показывает пользователю количество машин. Процесс работы сохраняется в видео.

2. Приложение : Бумажная клавиатура

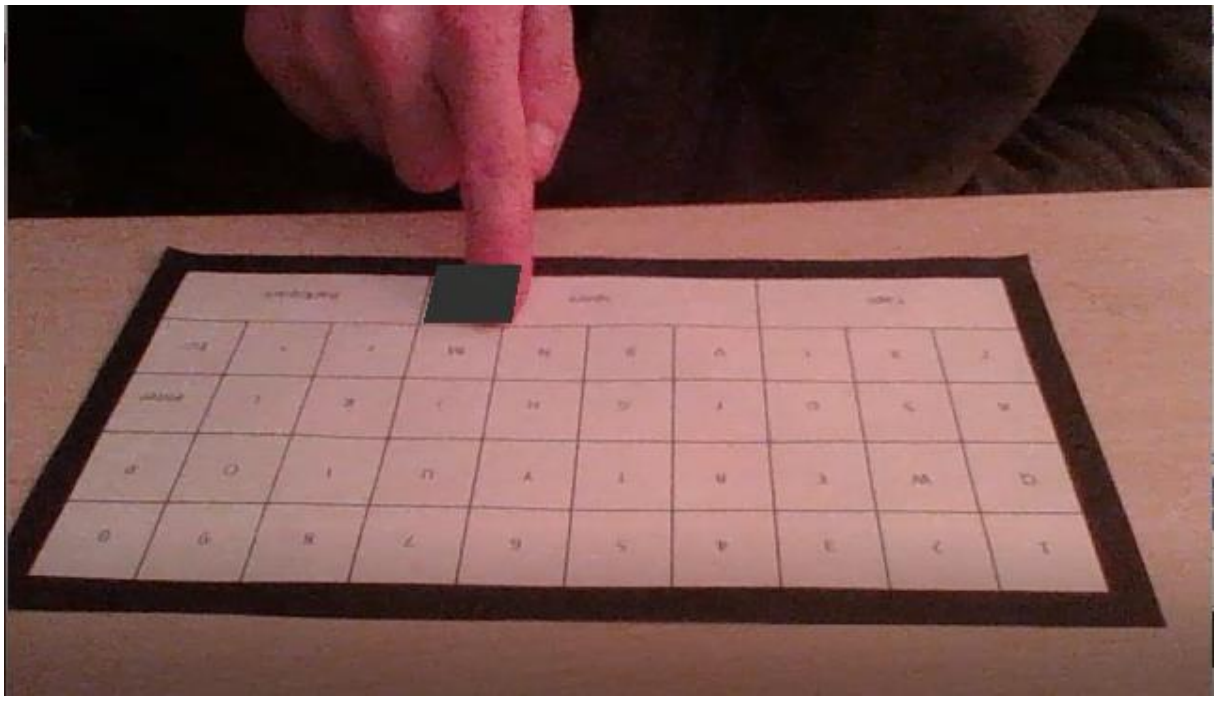
Образ клавиатуры:

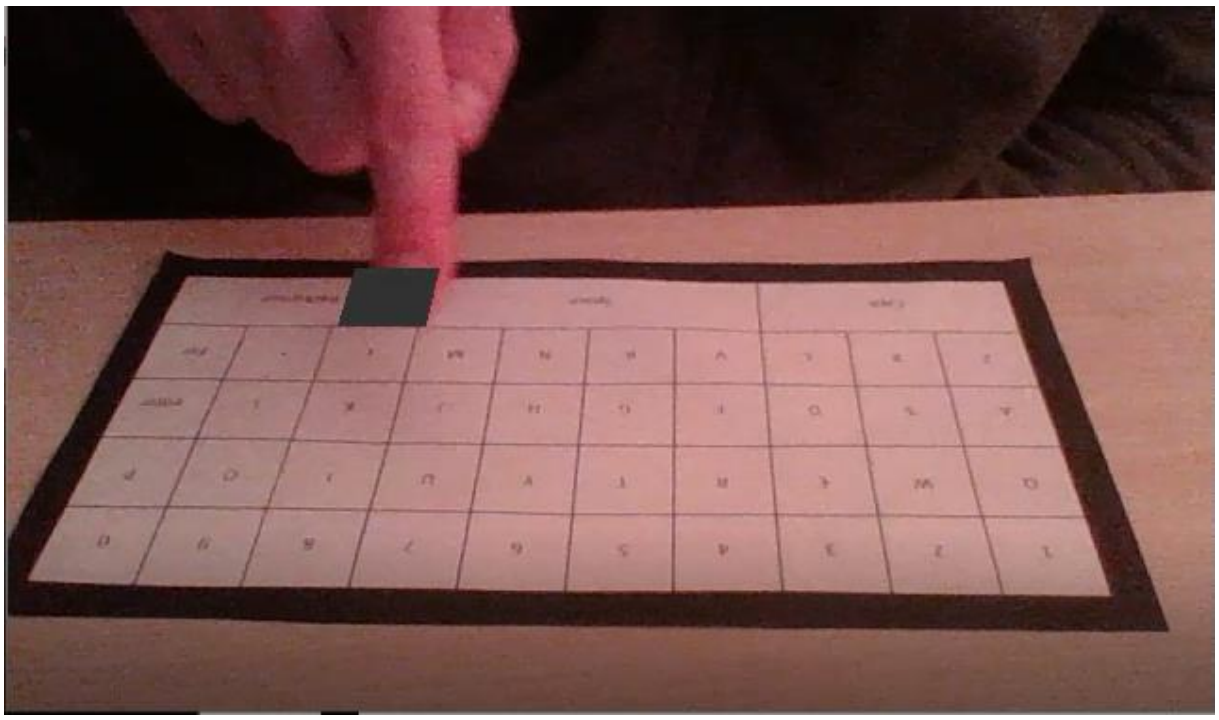


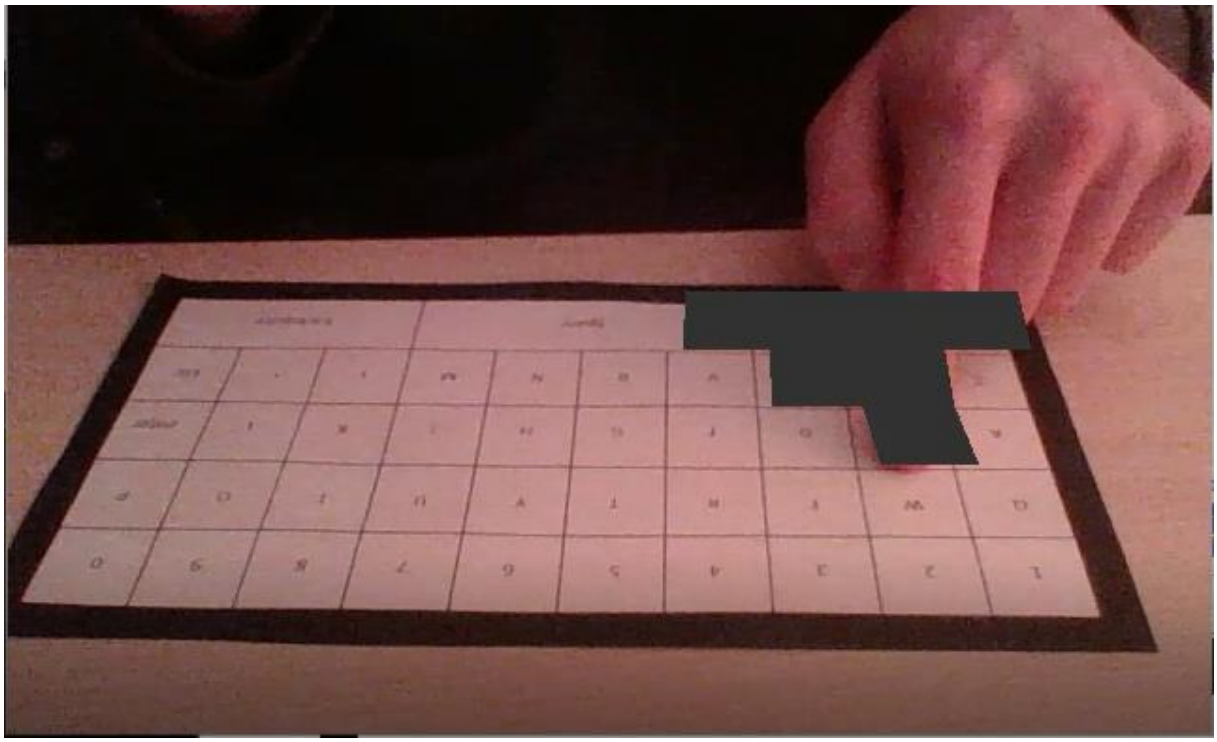
Этап калибрации:



Определение нажатой кнопки:







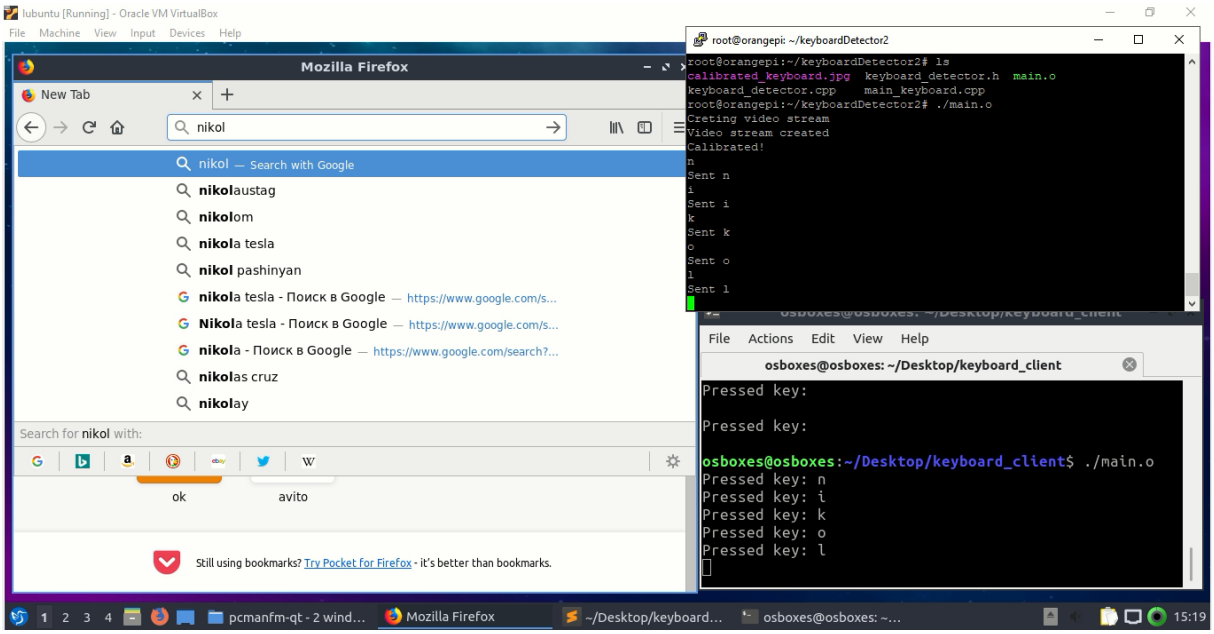
Компиляция и запуск программу:

```
# g++ -O2 -std=c++11 main_keyboard.cpp keyboard_detector.h
keyboard_detector.cpp `pkg-config opencv --cflags --libs` -o main.o


# ./main.o
```

Компиляция и запуск клиентского приложения:

```
# g++ client.cpp -lX11 -lXtst -o client.o


# ./client.o
```

Работа программы:



Заключение

В ходе работы был получен два модуля «Подсчёт машин» и «Бумажная клавиатура». Получили опыты работы с платой Orange Pi. Как подключить, работать на неё. Опыты работы с протоколом TCP/IP, с библиотекой Open CV на плате Orange Pi, а так же и с библиотекой Xlib.

Список литературы

1. https://github.com/96boards/documentation/blob/master/iot/orangepi-i96/hardware-docs/files/OrangePi%20i96%20User%20Manual_v0.9.1.pdf

2. https://www.cnx-software.com/2015/09/26/how-to-use-orange-pi-camera-in-linux-with-motion/

3. https://docs.opencv.org/2.4.13.7

4. https://tronche.com/gui/x/xlib/events/keyboard-pointer/keyboard-pointer.html

5. https://www.geeksforgeeks.org/socket-programming-cc/

Текст программы

1. Приложение: подсчёт машин

1.1 Blob.h

```
#pragma once

#include<opencv2/core/core.hpp>
#include<opencv2/highgui/highgui.hpp>
#include<opencv2/imgproc/imgproc.hpp>


class Blob // класс машины
{
public:
	std::vector<cv::Point> currentContour; // контур машины

	cv::Rect currentBoundingRect; // прямоугольник на этой машине

	std::vector<cv::Point> centerPositions; // центр этого прямоугольника

	double dblCurrentDiagonalSize; // диагональ этого прямоугольника
	double dblCurrentAspectRatio; // отношение между высотой и шириной

	bool blnCurrentMatchFoundOrNewBlob; // Текущее совпадение найдено или объект

	bool blnStillBeingTracked; // все еще отслеживается

	int intNumOfConsecutiveFramesWithoutAMatch; // количесто кадр

	cv::Point predictedNextPosition; // предсказать следующую позицию

	Blob(std::vector<cv::Point> _contour); // конструктор
	void predictNextPosition(void); // предсказать следующую позицию
};
```

1.2 Blob.cpp

```
#include "Blob.h"

Blob::Blob(std::vector<cv::Point> _contour) {

	currentContour = _contour;

	currentBoundingRect = cv::boundingRect(currentContour);

	cv::Point currentCenterOfRect;

	currentCenterOfRect.x = (currentBoundingRect.x + currentBoundingRect.x +
currentBoundingRect.width) / 2;
	currentCenterOfRect.y = (currentBoundingRect.y + currentBoundingRect.y +
currentBoundingRect.height) / 2;

	centerPositions.push_back(currentCenterOfRect);

	dblCurrentDiagonalSize = sqrt(pow(currentBoundingRect.width, 2) +
pow(currentBoundingRect.height, 2));

	dblCurrentAspectRatio = (float)currentBoundingRect.width /
(float)currentBoundingRect.height;
```

```
        blnStillBeingTracked = true;
        blnCurrentMatchFoundOrNewBlob = true;

        intNumOfConsecutiveFramesWithoutAMatch = 0;
}


void Blob::predictNextPosition(void) {

        int numPositions = (int)centerPositions.size();

        if (numPositions == 1) {

                predictedNextPosition.x = centerPositions.back().x;
                predictedNextPosition.y = centerPositions.back().y;

        }
        else if (numPositions == 2) {

                int deltaX = centerPositions[1].x - centerPositions[0].x;
                int deltaY = centerPositions[1].y - centerPositions[0].y;

                predictedNextPosition.x = centerPositions.back().x + deltaX;
                predictedNextPosition.y = centerPositions.back().y + deltaY;

        }
        else if (numPositions == 3) {

                int sumOfXChanges = ((centerPositions[2].x - centerPositions[1].x) * 2) +
                        ((centerPositions[1].x - centerPositions[0].x) * 1);

                int deltaX = (int)std::round((float)sumOfXChanges / 3.0);

                int sumOfYChanges = ((centerPositions[2].y - centerPositions[1].y) * 2) +
                        ((centerPositions[1].y - centerPositions[0].y) * 1);

                int deltaY = (int)std::round((float)sumOfYChanges / 3.0);

                predictedNextPosition.x = centerPositions.back().x + deltaX;
                predictedNextPosition.y = centerPositions.back().y + deltaY;

        }
        else if (numPositions == 4) {

                int sumOfXChanges = ((centerPositions[3].x - centerPositions[2].x) * 3) +
                        ((centerPositions[2].x - centerPositions[1].x) * 2) +
                        ((centerPositions[1].x - centerPositions[0].x) * 1);

                int deltaX = (int)std::round((float)sumOfXChanges / 6.0);

                int sumOfYChanges = ((centerPositions[3].y - centerPositions[2].y) * 3) +
                        ((centerPositions[2].y - centerPositions[1].y) * 2) +
                        ((centerPositions[1].y - centerPositions[0].y) * 1);

                int deltaY = (int)std::round((float)sumOfYChanges / 6.0);

                predictedNextPosition.x = centerPositions.back().x + deltaX;
                predictedNextPosition.y = centerPositions.back().y + deltaY;

        }
        else if (numPositions >= 5) {
```

```
            int sumOfXChanges = ((centerPositions[numPositions - 1].x -
centerPositions[numPositions - 2].x) * 4) +
                    ((centerPositions[numPositions - 2].x -
centerPositions[numPositions - 3].x) * 3) +
                    ((centerPositions[numPositions - 3].x -
centerPositions[numPositions - 4].x) * 2) +
                    ((centerPositions[numPositions - 4].x -
centerPositions[numPositions - 5].x) * 1);

            int deltaX = (int)std::round((float)sumOfXChanges / 10.0);

            int sumOfYChanges = ((centerPositions[numPositions - 1].y -
centerPositions[numPositions - 2].y) * 4) +
                    ((centerPositions[numPositions - 2].y -
centerPositions[numPositions - 3].y) * 3) +
                    ((centerPositions[numPositions - 3].y -
centerPositions[numPositions - 4].y) * 2) +
                    ((centerPositions[numPositions - 4].y -
centerPositions[numPositions - 5].y) * 1);

            int deltaY = (int)std::round((float)sumOfYChanges / 10.0);

            predictedNextPosition.x = centerPositions.back().x + deltaX;
            predictedNextPosition.y = centerPositions.back().y + deltaY;

        }
        else {
            // никогда ни будет
        }

}
```

1.3 main.cpp

```
#include <iostream>
#include <stdio.h>
#include <unistd.h>
#include <sys/socket.h>
#include <stdlib.h>
#include <netinet/in.h>
#include <string.h>
#include <arpa/inet.h>
#include <opencv2/highgui/highgui.hpp>
#include <opencv2/imgproc/imgproc.hpp>
#include "Blob.h"

#define SHOW_STEPS          //работает by step
#define PORT 8080           // адрес порта
// цвет для рисования результат на кадре
const cv::Scalar SCALAR_BLACK = cv::Scalar(0.0, 0.0, 0.0);
const cv::Scalar SCALAR_WHITE = cv::Scalar(255.0, 255.0, 255.0);
const cv::Scalar SCALAR_YELLOW = cv::Scalar(0.0, 255.0, 255.0);
const cv::Scalar SCALAR_GREEN = cv::Scalar(0.0, 200.0, 0.0);
const cv::Scalar SCALAR_RED = cv::Scalar(0.0, 0.0, 255.0);

void matchCurrentBlob(std::vector<Blob> &existingBlobs, std::vector<Blob>
&currentFrameBlobs); // соединить новый список объект в старый список объекта
```

```
void addBlobToLs(Blob &currentFrameBlob, std::vector<Blob> &existingBlobs, int
&intIndex);  // соединить новый объект в список объекта с позицией известной
void addNewBlob(Blob &currentFrameBlob, std::vector<Blob> &existingBlobs);
// соединить новый объект в конце списка объекта
double distance(cv::Point point1, cv::Point point2); // читает расстояние между точками
bool checkBlobCrossed(std::vector<Blob> &blobs, int &intHorizontalLinePosition, int
&carCount); // проверить машина пересекает ли дверь
void drawBlob(std::vector<Blob> &blobs, cv::Mat &imgFrame2Copy);
// отметить машину на картине
void drawCarCount(int &carCount, cv::Mat &imgFrame2Copy);
// рисовать результат на картине.

int main(void) {
	cv::VideoCapture capVideo;
	cv::Mat imgFrame1;
	cv::Mat imgFrame2;

	std::vector<Blob> blobs; // объект машины
	cv::Point crossingLine[2]; // отметить дверь
	int carCount = 0; // количество машин
	// создать сокет по адресу IP/Port
	struct sockaddr_in address;
	int sock = 0, valread;
	struct sockaddr_in serv_addr;
	char *message = (char*) "Car counting: ";
	char buffer[1024] = { 0 };
	if ((sock = socket(AF_INET, SOCK_STREAM, 0)) < 0)
	{
		printf("\n Socket creation error \n");
		return -1;
	}

	memset(&serv_addr, '0', sizeof(serv_addr));

	serv_addr.sin_family = AF_INET;
	serv_addr.sin_port = htons(PORT);

	// Convert IPv4 and IPv6 addresses from text to binary form
	if (inet_pton(AF_INET, "192.168.1.108", &serv_addr.sin_addr) <= 0)
	{
		printf("\nInvalid address/ Address not supported \n");
		return -1;
	}
	//подключает к серверу
	if (connect(sock, (struct sockaddr *)&serv_addr, sizeof(serv_addr)) < 0)
	{
		printf("\nConnection Failed \n");
		return -1;
	}
```

```
	capVideo.open("CarsDrivingUnderBridge.mp4"); // открыть видео для проверки
	//capVideo.open(0); // для окрыта камеры

	if (!capVideo.isOpened()) {                                 // если не удается окрыть Видео
		std::cout << "error reading video file" << std::endl << std::endl;      // выводит ошибку
		return(-1);
	}

	capVideo.read(imgFrame1);
	capVideo.read(imgFrame2);
	// видео сохранить результат
	int frame_w = capVideo.get(CV_CAP_PROP_FRAME_WIDTH);
	int frame_h = capVideo.get(CV_CAP_PROP_FRAME_HEIGHT);
	cv::VideoWriter outVideo("outVideo.avi", CV_FOURCC('M', 'J', 'P', 'G'), 10, cv::Size(frame_w, frame_h));

	int intHorizontalLinePosition = (int)std::round((double)imgFrame1.rows * 0.35);
	// линия рисованна вместо двери
	crossingLine[0].x = 0;
	crossingLine[0].y = intHorizontalLinePosition;

	crossingLine[1].x = imgFrame1.cols - 1;
	crossingLine[1].y = intHorizontalLinePosition;

	bool blnFirstFrame = true;

	int frameCount = 2;

	while (capVideo.isOpened()) {
		//while (chCheckForEscKey != 27) { // для работы с камерой
		std::vector<Blob> currentFrameBlobs;

		cv::Mat imgFrame1Copy = imgFrame1.clone();
		cv::Mat imgFrame2Copy = imgFrame2.clone();

		cv::Mat imgDifference;
		cv::Mat imgThresh;
		// предварительная обработка изображений
		cv::cvtColor(imgFrame1Copy, imgFrame1Copy, CV_BGR2GRAY);
		cv::cvtColor(imgFrame2Copy, imgFrame2Copy, CV_BGR2GRAY);

		cv::GaussianBlur(imgFrame1Copy, imgFrame1Copy, cv::Size(5, 5), 0);
		cv::GaussianBlur(imgFrame2Copy, imgFrame2Copy, cv::Size(5, 5), 0);

		cv::absdiff(imgFrame1Copy, imgFrame2Copy, imgDifference);

		cv::threshold(imgDifference, imgThresh, 30, 255.0, CV_THRESH_BINARY);
```

```
		cv::Mat structuringElement3x3 =
cv::getStructuringElement(cv::MORPH_RECT, cv::Size(3, 3));
		cv::Mat structuringElement5x5 =
cv::getStructuringElement(cv::MORPH_RECT, cv::Size(5, 5));
		cv::Mat structuringElement7x7 =
cv::getStructuringElement(cv::MORPH_RECT, cv::Size(7, 7));
		cv::Mat structuringElement15x15 =
cv::getStructuringElement(cv::MORPH_RECT, cv::Size(15, 15));

		for (unsigned int i = 0; i < 2; i++) {
			cv::dilate(imgThresh, imgThresh, structuringElement5x5);
			cv::dilate(imgThresh, imgThresh, structuringElement5x5);
			cv::erode(imgThresh, imgThresh, structuringElement5x5);
		}

		cv::Mat imgThreshCopy = imgThresh.clone();

		std::vector<std::vector<cv::Point> > contours;

		cv::findContours(imgThreshCopy, contours, cv::RETR_EXTERNAL,
cv::CHAIN_APPROX_SIMPLE);

		std::vector<std::vector<cv::Point> > convexHulls(contours.size());

		for (unsigned int i = 0; i < contours.size(); i++) {
			cv::convexHull(contours[i], convexHulls[i]);
		}


		for (auto &convexHull : convexHulls) {
			Blob possibleBlob(convexHull);
			// условие для проверки того что является ли машиной Blob
			if (possibleBlob.currentBoundingRect.area() > 400 &&
				possibleBlob.dblCurrentAspectRatio > 0.2 &&
				possibleBlob.dblCurrentAspectRatio < 4.0 &&
				possibleBlob.currentBoundingRect.width > 30 &&
				possibleBlob.currentBoundingRect.height > 30 &&
				possibleBlob.dblCurrentDiagonalSize > 60.0 &&
				(cv::contourArea(possibleBlob.currentContour) /
(double)possibleBlob.currentBoundingRect.area()) > 0.50) {
				currentFrameBlobs.push_back(possibleBlob);
			}
		}


		if (blnFirstFrame == true) {
			for (auto &currentFrameBlob : currentFrameBlobs) {
				blobs.push_back(currentFrameBlob);
			}
		}
```

```
                else {
                        matchCurrentBlob(blobs, currentFrameBlobs);
                }

                imgFrame2Copy = imgFrame2.clone();      // работаем на копи-версиях

                drawBlob(blobs, imgFrame2Copy);
                // Проверяет что машина прошла ли линию перехода
                bool blnAtLeastOneBlobCrossedTheLine = checkBlobCrossed(blobs,
intHorizontalLinePosition, carCount);

                if (blnAtLeastOneBlobCrossedTheLine == true) {
                        cv::line(imgFrame2Copy, crossingLine[0], crossingLine[1],
SCALAR_GREEN, 2);
                }
                else {
                        cv::line(imgFrame2Copy, crossingLine[0], crossingLine[1],
SCALAR_RED, 2);
                }

                drawCarCount(carCount, imgFrame2Copy);

                outVideo.write(imgFrame2Copy);
                //передать результат carCount
                std::string s = std::to_string(carCount);
                char const *pchar = s.c_str();
                send(sock, pchar, strlen(pchar), 0);
                printf("Sent %s \n", pchar);
                //cv::waitKey(0);               // работает by step

                currentFrameBlobs.clear();

                imgFrame1 = imgFrame2.clone();

                if ((capVideo.get(CV_CAP_PROP_POS_FRAMES) + 1) <
capVideo.get(CV_CAP_PROP_FRAME_COUNT)) {
                        capVideo.read(imgFrame2);
                }
                else {
                        std::cout << "end of video\n";
                        break;
                }
                capVideo.read(imgFrame2);
                blnFirstFrame = false;
                frameCount++;
        }
        return(0);
}
```

```
void matchCurrentBlob(std::vector<Blob> &existingBlobs, std::vector<Blob>
&currentFrameBlobs) {

	for (auto &existingBlob : existingBlobs) {

		existingBlob.blnCurrentMatchFoundOrNewBlob = false;

		existingBlob.predictNextPosition();
	}

	for (auto &currentFrameBlob : currentFrameBlobs) {

		int intIndexOfLeastDistance = 0;
		double dblLeastDistance = 100000.0;

		for (unsigned int i = 0; i < existingBlobs.size(); i++) {

			if (existingBlobs[i].blnStillBeingTracked == true) {

				double dblDistance =
distance(currentFrameBlob.centerPositions.back(), existingBlobs[i].predictedNextPosition);

				if (dblDistance < dblLeastDistance) {
					dblLeastDistance = dblDistance;
					intIndexOfLeastDistance = i;
				}
			}
		}

		if (dblLeastDistance < currentFrameBlob.dblCurrentDiagonalSize * 0.5) {
			addBlobToLs(currentFrameBlob, existingBlobs,
intIndexOfLeastDistance);
		}
		else {
			addNewBlob(currentFrameBlob, existingBlobs);
		}

	}

	for (auto &existingBlob : existingBlobs) {

		if (existingBlob.blnCurrentMatchFoundOrNewBlob == false) {
			existingBlob.intNumOfConsecutiveFramesWithoutAMatch++;
		}

		if (existingBlob.intNumOfConsecutiveFramesWithoutAMatch >= 5) {
			existingBlob.blnStillBeingTracked = false;
		}

	}
```

```
}

void addBlobToLs(Blob &currentFrameBlob, std::vector<Blob> &existingBlobs, int
&intIndex) {

	existingBlobs[intIndex].currentContour = currentFrameBlob.currentContour;
	existingBlobs[intIndex].currentBoundingRect =
currentFrameBlob.currentBoundingRect;

	existingBlobs[intIndex].centerPositions.push_back(currentFrameBlob.centerPositions.
back());

	existingBlobs[intIndex].dblCurrentDiagonalSize =
currentFrameBlob.dblCurrentDiagonalSize;
	existingBlobs[intIndex].dblCurrentAspectRatio =
currentFrameBlob.dblCurrentAspectRatio;

	existingBlobs[intIndex].blnStillBeingTracked = true;
	existingBlobs[intIndex].blnCurrentMatchFoundOrNewBlob = true;
}

void addNewBlob(Blob &currentFrameBlob, std::vector<Blob> &existingBlobs) {

	currentFrameBlob.blnCurrentMatchFoundOrNewBlob = true;

	existingBlobs.push_back(currentFrameBlob);
}

double distance(cv::Point point1, cv::Point point2) {

	int intX = abs(point1.x - point2.x);
	int intY = abs(point1.y - point2.y);

	return(sqrt(pow(intX, 2) + pow(intY, 2)));
}

bool checkBlobCrossed(std::vector<Blob> &blobs, int &intHorizontalLinePosition, int
&carCount) {
	bool blnAtLeastOneBlobCrossedTheLine = false;

	for (auto blob : blobs) {

		if (blob.blnStillBeingTracked == true && blob.centerPositions.size() >= 2) {
			int prevFrameIndex = (int)blob.centerPositions.size() - 2;
			int currFrameIndex = (int)blob.centerPositions.size() - 1;

			if (blob.centerPositions[prevFrameIndex].y > intHorizontalLinePosition
&& blob.centerPositions[currFrameIndex].y <= intHorizontalLinePosition) {
				carCount++;
```

```
                        blnAtLeastOneBlobCrossedTheLine = true;
                    }
                }

            }

            return blnAtLeastOneBlobCrossedTheLine;
}

void drawBlob(std::vector<Blob> &blobs, cv::Mat &imgFrame2Copy) {

            for (unsigned int i = 0; i < blobs.size(); i++) {

                if (blobs[i].blnStillBeingTracked == true) {
                    cv::rectangle(imgFrame2Copy, blobs[i].currentBoundingRect,
SCALAR_RED, 2);

                    int intFontFace = CV_FONT_HERSHEY_SIMPLEX;
                    double dblFontScale = blobs[i].dblCurrentDiagonalSize / 60.0;
                    int intFontThickness = (int)std::round(dblFontScale * 1.0);

                    cv::putText(imgFrame2Copy, std::to_string(i),
blobs[i].centerPositions.back(), intFontFace, dblFontScale, SCALAR_GREEN,
intFontThickness);
                }
            }
}


void drawCarCount(int &carCount, cv::Mat &imgFrame2Copy) {

            int intFontFace = CV_FONT_HERSHEY_SIMPLEX;
            double dblFontScale = (imgFrame2Copy.rows * imgFrame2Copy.cols) / 300000.0;
            int intFontThickness = (int)std::round(dblFontScale * 1.5);

            cv::Size textSize = cv::getTextSize(std::to_string(carCount), intFontFace,
dblFontScale, intFontThickness, 0);

            cv::Point ptTextBottomLeftPosition;

            ptTextBottomLeftPosition.x = imgFrame2Copy.cols - 1 - (int)((double)textSize.width
* 1.25);
            ptTextBottomLeftPosition.y = (int)((double)textSize.height * 1.25);

            cv::putText(imgFrame2Copy, std::to_string(carCount), ptTextBottomLeftPosition,
intFontFace, dblFontScale, SCALAR_GREEN, intFontThickness);

}
```

1.4 client.cpp

```
#include <unistd.h>
#include <stdio.h>
#include <sys/socket.h>
#include <stdlib.h>
#include <netinet/in.h>
#include <string.h>
#define PORT 8080
int main(int argc, char const *argv[])
{
   int server_fd, new_socket, valread;
   struct sockaddr_in address;
   int opt = 1;
   int addrlen = sizeof(address);
   char buffer[1024] = {0};
   //char *hello = (char*)"Hello from server";

   // Creating socket file descriptor
   if ((server_fd = socket(AF_INET, SOCK_STREAM, 0)) == 0)
   {
      perror("socket failed");
      exit(EXIT_FAILURE);
   }

   // Forcefully attaching socket to the port 8080
   if (setsockopt(server_fd, SOL_SOCKET, SO_REUSEADDR | SO_REUSEPORT,
                                    &opt, sizeof(opt)))
   {
      perror("setsockopt");
      exit(EXIT_FAILURE);
   }
   address.sin_family = AF_INET;
   address.sin_addr.s_addr = INADDR_ANY;
   address.sin_port = htons( PORT );

   // Forcefully attaching socket to the port 8080
   if (bind(server_fd, (struct sockaddr *)&address,
                           sizeof(address))<0)
   {
      perror("bind failed");
      exit(EXIT_FAILURE);
   }
   if (listen(server_fd, 3) < 0)
   {
      perror("listen");
      exit(EXIT_FAILURE);
   }
   if ((new_socket = accept(server_fd, (struct sockaddr *)&address,
                  (socklen_t*)&addrlen))<0)
   {
```

```
        perror("accept");
        exit(EXIT_FAILURE);
    }
    int count = 0;
    valread = 1;
    while (valread > 0) {
          valread = read( new_socket , buffer, 1024);
          printf("Car counting: %s \n",buffer );
          count ++;
    }
    //send(new_socket , hello , strlen(hello) , 0 );
    //printf("Hello message sent\n");
    return 0;
}
```

2. Приложение: бумажная клавиатура

2.1 keyboard_detector.h

```
#ifndef KEYBOARD_DETECTOR
#define KEYBOARD_DETECTOR

#include "opencv2/imgproc/imgproc.hpp"
#include "opencv2/highgui/highgui.hpp"
#include <iostream>
#include <vector>
#include <algorithm>
#include <functional>

using namespace cv;
using namespace std;
class KeyboardDetector
{

public:

  void start();

private:

  Mat keyboardRegion, binKeyboard;
  const int horizontalPoints = 11;
  const int verticalPoints = 6;
  Point keyboardPoints[6][11];
  uchar medianKeyValues[5][10];
  char keySymbols[5][10] = { {'<','<','<',' ',' ',' ',' ',' ','^','^','^'},
                             {'E','.',',','m','n','b','v','c','x','z'},
                             {'\n','l','k','j','h','g','f','d','s','a'},
```

```
                {'p','o','i','u','y','t','r','e','w','q'},
                {'0','9','8','7','6','5','4','3','2','1'} };

  int getKeyboardContourIdx(vector<vector<Point>> &contours);

  void getKeyboardVertices(vector<vector<Point>> &contours, int index, vector<Point>
&vertices);

  void drawKeyboardLines(Mat &dst, vector<Point> &vertices);

  void keyboardVertexCorrection(vector<Point> &vertices);

  void setupBinKeyboard();

  int getMedianPixel(Mat &input);

  bool isKeyPressed(int row, int col, Mat &gray);

  void rotateAroundPoint(Point &p, Point pivot, double angle);
};

#endif
```

2.2 keyboard_detector.cpp

```
#include "keyboard_detector.h"
#include <iostream>
#include <stdio.h>
#include <unistd.h>
#include <sys/socket.h>
#include <stdlib.h>
#include <netinet/in.h>
#include <string.h>
#include <arpa/inet.h>

#define PORT 8080
#define CLIENT_ADDR "192.168.43.40"
#define EPS 0.000001

void KeyboardDetector::start()
{
        struct sockaddr_in address;
        int sock = 0, valread;
        struct sockaddr_in serv_addr;
        char *message = (char*) "Car counting: ";
        char buffer[1024] = {0};
        if ((sock = socket(AF_INET, SOCK_STREAM, 0)) < 0)
        {
        printf("\n Socket creation error \n");
        return;
```

```
        }

        memset(&serv_addr, '0', sizeof(serv_addr));

        serv_addr.sin_family = AF_INET;
        serv_addr.sin_port = htons(PORT);

        // Convert IPv4 and IPv6 addresses from text to binary form
        if(inet_pton(AF_INET, CLIENT_ADDR, &serv_addr.sin_addr)<=0)
        {
        printf("\nInvalid address/ Address not supported \n");
        return;
        }

        if (connect(sock, (struct sockaddr *)&serv_addr, sizeof(serv_addr)) < 0)
        {
        printf("\nConnection Failed \n");
        return;
        }



 using namespace cv;

 std::cout << "Creting video stream" << std::endl;
 VideoCapture stream(0);
 std::cout << "Video stream created" << std::endl;

 if (!stream.isOpened())
 {
  std::cout << "Error: Cannot open video capture stream!" << std::endl;
  return;
 }
 stream.set(CV_CAP_PROP_FRAME_WIDTH, 640);
 stream.set(CV_CAP_PROP_FRAME_HEIGHT, 480);

 while (true)
 {
  Mat frame;
  stream.read(frame);
  Mat gray;
  cvtColor(frame, gray, COLOR_BGR2GRAY);

  std::vector<std::vector<Point>> contours;
  Mat bin;
  adaptiveThreshold(gray, bin, 255, ADAPTIVE_THRESH_MEAN_C,
THRESH_BINARY_INV, 13, 2);
  findContours(bin, contours, CV_RETR_EXTERNAL, CV_CHAIN_APPROX_SIMPLE);

  if (contours.empty())
```

```
    continue;

  Mat drawing = frame;
  int maxIdx = getKeyboardContourIdx(contours);
  std::vector<Point> vertices;
  getKeyboardVertices(contours, maxIdx, vertices);

  keyboardRegion = frame(boundingRect(contours[maxIdx]));

  setupBinKeyboard();
  keyboardVertexCorrection(vertices);
  drawKeyboardLines(drawing, vertices);

  //calculate median pixel value in all cells
  for (int i = 0; i < verticalPoints - 1; i++)
  {
    for (int j = 0; j < horizontalPoints - 1; j++)
    {
      Rect keyRect = Rect(keyboardPoints[i][j], keyboardPoints[i + 1][j + 1]);
      Mat keyRegion = gray(keyRect);
      medianKeyValues[i][j] = getMedianPixel(keyRegion);
    }
  }

  //try to finish calibration

  bool isCalibrated = true;
  for (int i = 0; i < 10; i++)
  {
    Mat frame;
    stream.read(frame);
    Mat gray;
    cvtColor(frame, gray, COLOR_BGR2GRAY);

    std::vector<std::vector<Point>> contours;
    Mat bin;
    adaptiveThreshold(gray, bin, 255, ADAPTIVE_THRESH_MEAN_C,
THRESH_BINARY_INV, 13, 2);
    findContours(bin, contours, CV_RETR_EXTERNAL, CV_CHAIN_APPROX_SIMPLE);

    if (contours.empty())
      continue;

    Mat drawing = frame;
    int maxIdx = getKeyboardContourIdx(contours);
    std::vector<Point> vertices;
    getKeyboardVertices(contours, maxIdx, vertices);

    keyboardRegion = frame(boundingRect(contours[maxIdx]));
```

```
  setupBinKeyboard();
  keyboardVertexCorrection(vertices);
  drawKeyboardLines(drawing, vertices);


  //imshow("img", drawing);


  //calculate median pixel value in all cells

  for (int i = 0; i < verticalPoints - 1; i++)
  {
    for (int j = 0; j < horizontalPoints - 1; j++)
    {
      Rect keyRect = Rect(keyboardPoints[i][j], keyboardPoints[i + 1][j + 1]);
      Mat keyRegion = gray(keyRect);

      if (abs(medianKeyValues[i][j] - getMedianPixel(keyRegion)) > 5)
      {
        isCalibrated = false;
        break;
      }
    }
    if (!isCalibrated)
      break;
  }

}



if (isCalibrated)
{
  std::cout << "Calibrated!" << std::endl;
  imwrite("calibrated_keyboard.jpg", drawing);
  while (true)
  {

    Mat frame;
    stream.read(frame);
    Mat gray;
    cvtColor(frame, gray, COLOR_BGR2GRAY);

    bool keysPressed[5][10] = { false };
    bool isAKeyPressed = false;
    for (int i = 0; i < verticalPoints - 1; i++)
    {
      for (int j = 0; j < horizontalPoints - 1; j++)
      {
        if (isKeyPressed(i, j, gray))
```

```
    {
      //fill in key
      vector<Point> keyPoints(4);
      keyPoints[0] = keyboardPoints[i][j];
      keyPoints[1] = keyboardPoints[i][j + 1];
      keyPoints[2] = keyboardPoints[i + 1][j + 1];
      keyPoints[3] = keyboardPoints[i + 1][j];

      fillConvexPoly(frame, keyPoints, Scalar(50, 50, 50));
      keysPressed[i][j] = true;
      isAKeyPressed = true;
    }
  }
}

//imshow("pressedKeys", frame);

static char lastKeyPressed;
static int lastKeyRow = -1, lastKeyCol = -1;
static bool lastKeyPrinted = true;

if (!isAKeyPressed && !lastKeyPrinted)
{
  std::cout << lastKeyPressed << std::endl;

              std::string s = "";
              s.push_back(lastKeyPressed);
              char const *pchar = s.c_str();
              send(sock , pchar , strlen(pchar) , 0 );
              printf("Sent %s \n", pchar);

  lastKeyPrinted = true;
  lastKeyRow = -1;
}
else
{
  for (int i = verticalPoints - 2; i >= 0; i--)
  {
    bool topKeyFound = false;
    for (int j = horizontalPoints - 2; j >= 0; j--)
    {
      if (keysPressed[i][j])
      {
        //std::cout << keySymbols[i][j];
        if (i > lastKeyRow || lastKeyRow == -1)
        {
          lastKeyRow = i;
          lastKeyPressed = keySymbols[i][j];
          lastKeyPrinted = false;
        }
```

```
            topKeyFound = true;
            break;
           }
          }
          if (topKeyFound)
           break;
         }
        }
      }
    }

}

int KeyboardDetector::getKeyboardContourIdx(vector<vector<Point>>& contours)
{
 int maxIdx = 0;
 double maxArea = contourArea(contours[0]);
 for (int i = 0; i < contours.size(); i++)
 {
  double currentArea = contourArea(contours[i]);
  if (currentArea > maxArea)
  {
   maxIdx = i;
   maxArea = currentArea;
  }
 }
 return maxIdx;
}

void KeyboardDetector::getKeyboardVertices(vector<vector<Point>>& contours, int index,
vector<Point>& vertices)
{
 Rect r = boundingRect(contours[index]);
 double EpsH = r.height / 10;
 int upperY = r.y;
 int leftIdx = -1, rightIdx = -1;

 for (int i = 0; i < contours[index].size(); i++)
 {
  if (std::abs(contours[index][i].y - upperY) < EpsH)
  {
   if (leftIdx == -1 || contours[index][i].x < contours[index][leftIdx].x)
   {
    leftIdx = i;
   }
   if (rightIdx == -1 || contours[index][i].x > contours[index][rightIdx].x)
   {
    rightIdx = i;
   }
```

```
    }
  }

  vertices.resize(4);
  vertices[0] = Point(contours[index][leftIdx].x, upperY);
  vertices[1] = Point(contours[index][rightIdx].x, upperY);
  vertices[2] = Point(r.x, r.y + r.height);
  vertices[3] = Point(r.x + r.width, r.y + r.height);
}

void KeyboardDetector::drawKeyboardLines(Mat & dst, vector<Point>& vertices)
{
  const int keysInRow = 10, keysInCol = 5;

  int w1 = vertices[1].x - vertices[0].x;
  int dx1 = w1 / keysInRow;
  int w2 = vertices[3].x - vertices[2].x;
  int dx2 = w2 / keysInRow;

  //Draw vertical lines:
  for (int i = 0; i <= keysInRow; i++)
  {
    line(dst, Point(vertices[0].x + i * dx1, vertices[0].y),
      Point(vertices[2].x + i * dx2, vertices[2].y), Scalar(0, 0, 255));
  }

  int h1 = vertices[2].y - vertices[0].y;
  int dy1 = h1 / keysInCol;
  double k1 = (vertices[0].y - vertices[2].y) / (vertices[0].x - vertices[2].x + EPS);
  double k2 = (vertices[1].y - vertices[3].y) / (vertices[1].x - vertices[3].x + EPS);

  //Draw horizontal lines:
  double dy_sameHorizontal = vertices[0].y - vertices[1].y;
  double dy_inLine = dy_sameHorizontal / (double)keysInRow;


  for (int i = 0; i <= keysInCol; i++)
  {
    int y = vertices[0].y + i * dy1 - (double)i*0.01*h1;

    int xLeft = (double)(y - vertices[0].y + k1 * vertices[0].x) / k1;
    int xRight = (double)(y - vertices[1].y + k2 * vertices[1].x) / k2;
    line(dst, Point(xLeft, y), Point(xRight, y), Scalar(0, 0, 255));

    int yOriginal = y;
    //draw intersection points
    int dx = (xRight - xLeft) / keysInRow;
    for (int j = 0; j <= keysInRow; j++)
    {
      int x = xLeft + j * dx;
```

```
      keyboardPoints[i][j] = Point(x, y);
    }
  }

  for (int i = 0; i <= keysInCol; i++)
    for (int j = 0; j <= keysInRow; j++)
      circle(dst, keyboardPoints[i][j], 3, Scalar(0, 0, 255), -1);
}

void KeyboardDetector::keyboardVertexCorrection(vector<Point> &vertices)
{
  vector<vector<Point>> innerContours;
  findContours(binKeyboard, innerContours, CV_RETR_EXTERNAL,
CV_CHAIN_APPROX_SIMPLE);
  vector<Point> innerVertices;
  getKeyboardVertices(innerContours, getKeyboardContourIdx(innerContours),
innerVertices);

  Size roiSize;
  Point offset;
  keyboardRegion.locateROI(roiSize, offset);
  vertices.resize(4);
  for (int i = 0; i < innerVertices.size(); i++)
  {
    vertices[i] = innerVertices[i] + offset;
  }
}

void KeyboardDetector::setupBinKeyboard()
{
  Mat keyboardRegionGray;
  cvtColor(keyboardRegion, keyboardRegionGray, COLOR_BGR2GRAY);

  threshold(keyboardRegionGray, binKeyboard, 200, 255, THRESH_OTSU |
THRESH_BINARY);
  dilate(binKeyboard, binKeyboard, Mat());
  erode(binKeyboard, binKeyboard, Mat());
}

int KeyboardDetector::getMedianPixel(Mat & input)
{
  vector<uchar> pixels;
  pixels.reserve(input.rows*input.cols);
  for (int i = 0; i < input.rows; i++)
    for (int j = 0; j < input.cols; j++)
      pixels.push_back(input.at<uchar>(i, j));

  sort(pixels.begin(), pixels.end());
  return pixels[pixels.size() / 2];
```

```
}

bool KeyboardDetector::isKeyPressed(int row, int col, Mat & gray)
{
  Rect keyRect = Rect(keyboardPoints[row][col], keyboardPoints[row + 1][col + 1]);
  Mat keyRegion = gray(keyRect);

  uchar medianPixel = getMedianPixel(keyRegion);

  if (abs(medianPixel - medianKeyValues[row][col]) < 35)
  {
    return false;
  }
  return true;

}

void KeyboardDetector::rotateAroundPoint(Point & p, Point pivot, double angle)
{
  double s = sin(angle);
  double c = cos(angle);

  // translate point back to origin:
  p.x -= pivot.x;
  p.y -= pivot.y;

  // rotate point
  float xnew = p.x * c - p.y * s;
  float ynew = p.x * s + p.y * c;

  // translate point back:
  p.x = xnew + pivot.x;
  p.y = ynew + pivot.y;
}
```

2.3 keyboard_main.cpp

```
#include <iostream>
#include "keyboard_detector.h"

int main()
{
  KeyboardDetector d;
  d.start();

  return 0;
}
```

2.4 client.cpp

```
#include <unistd.h>
#include <stdio.h>
#include <sys/socket.h>
#include <stdlib.h>
#include <netinet/in.h>
#include <string.h>

#include <X11/Xlib.h>
#include <X11/keysym.h>
#include <X11/extensions/XTest.h>

#define PORT 8080

int main(int argc, char const *argv[])
{
        Display *display;
        unsigned int keycode;
        display = XOpenDisplay(NULL);


   int server_fd, new_socket, valread;
   struct sockaddr_in address;
   int opt = 1;
   int addrlen = sizeof(address);
   char buffer[1024] = {0};

   // Creating socket file descriptor
   if ((server_fd = socket(AF_INET, SOCK_STREAM, 0)) == 0)
   {
      perror("socket failed");
      exit(EXIT_FAILURE);
   }

   // Forcefully attaching socket to the port 8080
   if (setsockopt(server_fd, SOL_SOCKET, SO_REUSEADDR | SO_REUSEPORT,
                                    &opt, sizeof(opt)))
   {
      perror("setsockopt");
      exit(EXIT_FAILURE);
   }
   address.sin_family = AF_INET;
   address.sin_addr.s_addr = INADDR_ANY;
   address.sin_port = htons( PORT );

   // Forcefully attaching socket to the port 8080
   if (bind(server_fd, (struct sockaddr *)&address,
                        sizeof(address))<0)
   {
      perror("bind failed");
```

```
    exit(EXIT_FAILURE);
  }
  if (listen(server_fd, 3) < 0)
  {
    perror("listen");
    exit(EXIT_FAILURE);
  }
  if ((new_socket = accept(server_fd, (struct sockaddr *)&address,
                (socklen_t*)&addrlen))<0)
  {
    perror("accept");
    exit(EXIT_FAILURE);
  }
  int count = 0;
  valread = 1;
  while (valread > 0)
  {
       valread = read( new_socket , buffer, 1024);
       printf("Pressed key: %s \n",buffer );
                count ++;


                char pressedKey = buffer[0];

                switch(pressedKey)
                {
                 case 'E':
                        keycode = XKeysymToKeycode(display, XK_Escape);
                        break;
                 case '<':
                        keycode = XKeysymToKeycode(display, XK_BackSpace);
                        break;
                 case '^':
                        keycode = XKeysymToKeycoe(display, XK_Caps_Lock);
                        break;
                 case '\n':
                        keycode = XKeysymToKeycode(display, XK_Return);
                        break;
                 default:
                        keycode = XKeysymToKeycode(display, pressedKey);
                        break;
                }

                XTestFakeKeyEvent(display, keycode, True, 0);
                XTestFakeKeyEvent(display, keycode, False, 0);
                XFlush(display);

  }
  return 0;
}
```