AlphaBot2

Сервис раздел

- Wiki-страница с общей информацией о роботе.
- Принципиальная схема платы-шасси робота.
- Принципиальная схема платы-адаптера робота.
- User manual
- Набор документации на компоненты робота
- Примеры кода `wget https://www.waveshare.com/w/upload/e/ee/AlphaBot2-Demo.7z`

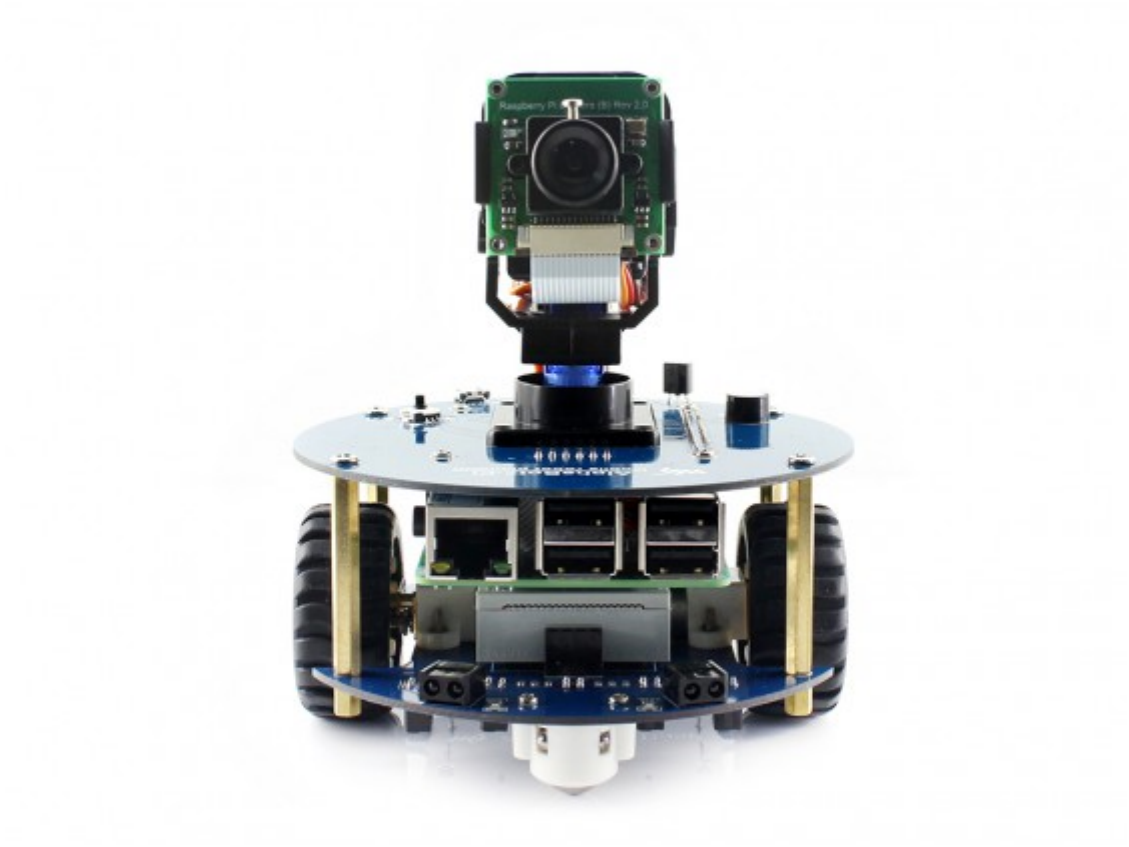
Скачивание и разархивированиеы:

```
cd
wget https://www.waveshare.com/w/upload/7/74/AlphaBot2.tar.gz
tar zxvf AlphaBot2.tar.gz
```

Для того чтобы смонтировать директорию с rpi себе на комп нужно следующее:

```
mkdir -p /tmp/server
sshfs pi@raspberrypi:/home/pi/alphabot2 /tmp/server
code /tmp/server
```

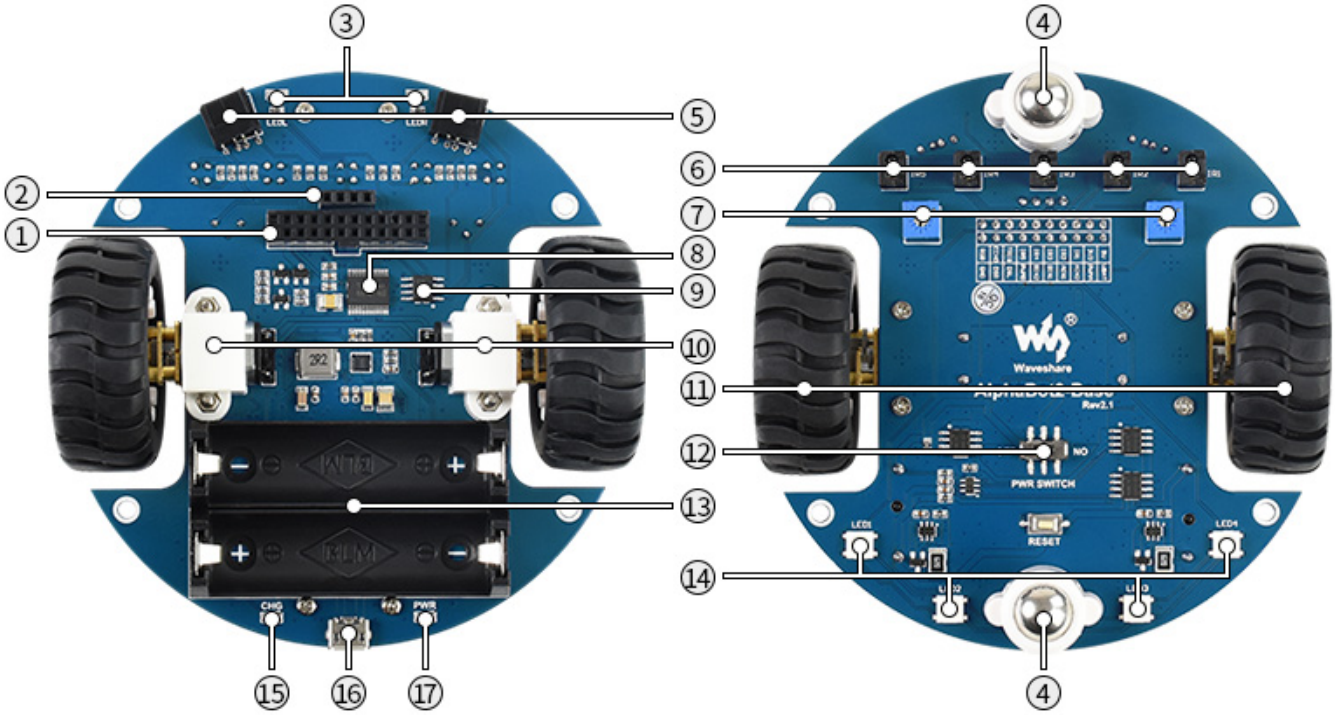
Знакомство с роботом AlphaBot2-Pi



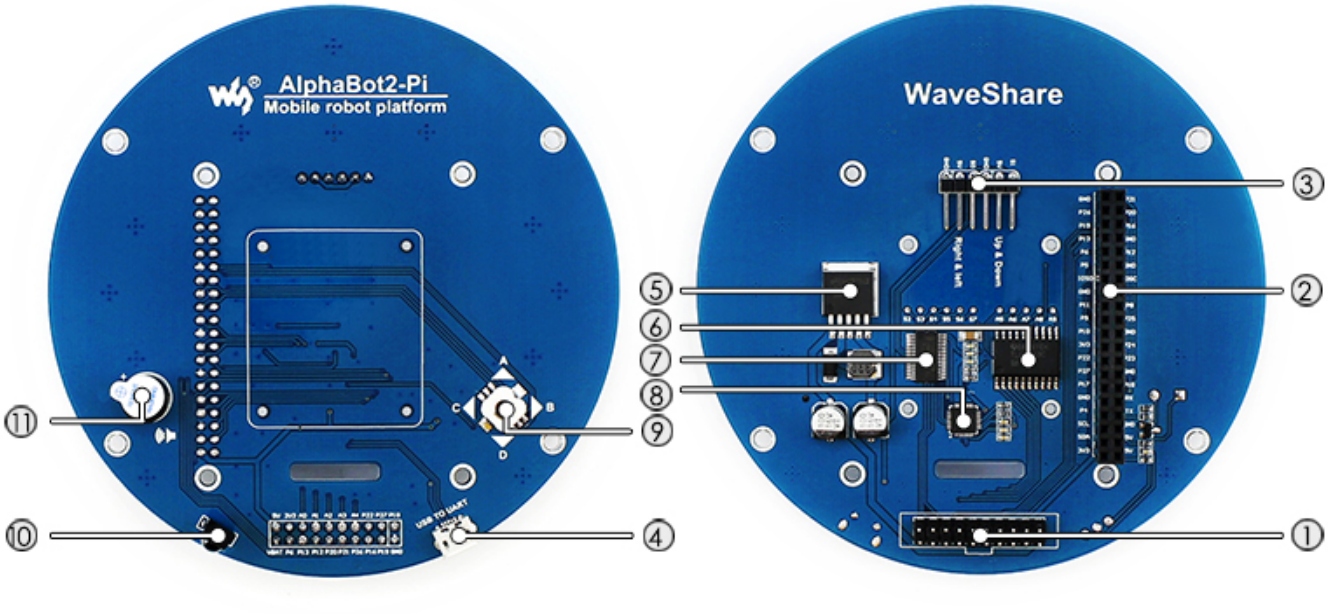
Робот AlphaBot2-Pi состоит из нескольких основных частей:

- База-шасси
- Плата-адаптер
- Одноплатный компьютер Raspberry-Pi
- Камера с поворотным кронштейном
- Ультразвуковой дальномер

База-шасси



1. Интерфейс управления шасси для подключения платы-адаптера.
2. Интерфейс для подключение ультразвукового дальномера.
3. Светодиодные индикаторы объезда препятствий.
4. Шаровая опора.
5. ST188 - инфракрасный фотоэлектрический датчик для объезда препятствий.
6. ITR20001/T - инфракрасный фотоэлектрический датчик для обнаружения линии.
7. Потенциометр для настройки дистанции объезда препятствий.
8. TB6612FNG - драйвер для моторов на H-мосте.
9. LM393 - компаратор напряжения.
10. N20 - мотор-редуктор, 1:30, 6V/600RPM.
11. Колеса диаметром 42мм, шириной 19мм.
12. Переключатель питания.
13. Разъем для двух батарей 14500.
14. WS2812B - RGB светодиоды.
15. Индикатор зарядки батареи.
16. Порт зарядки USB 5V.
17. Индикатор питания.



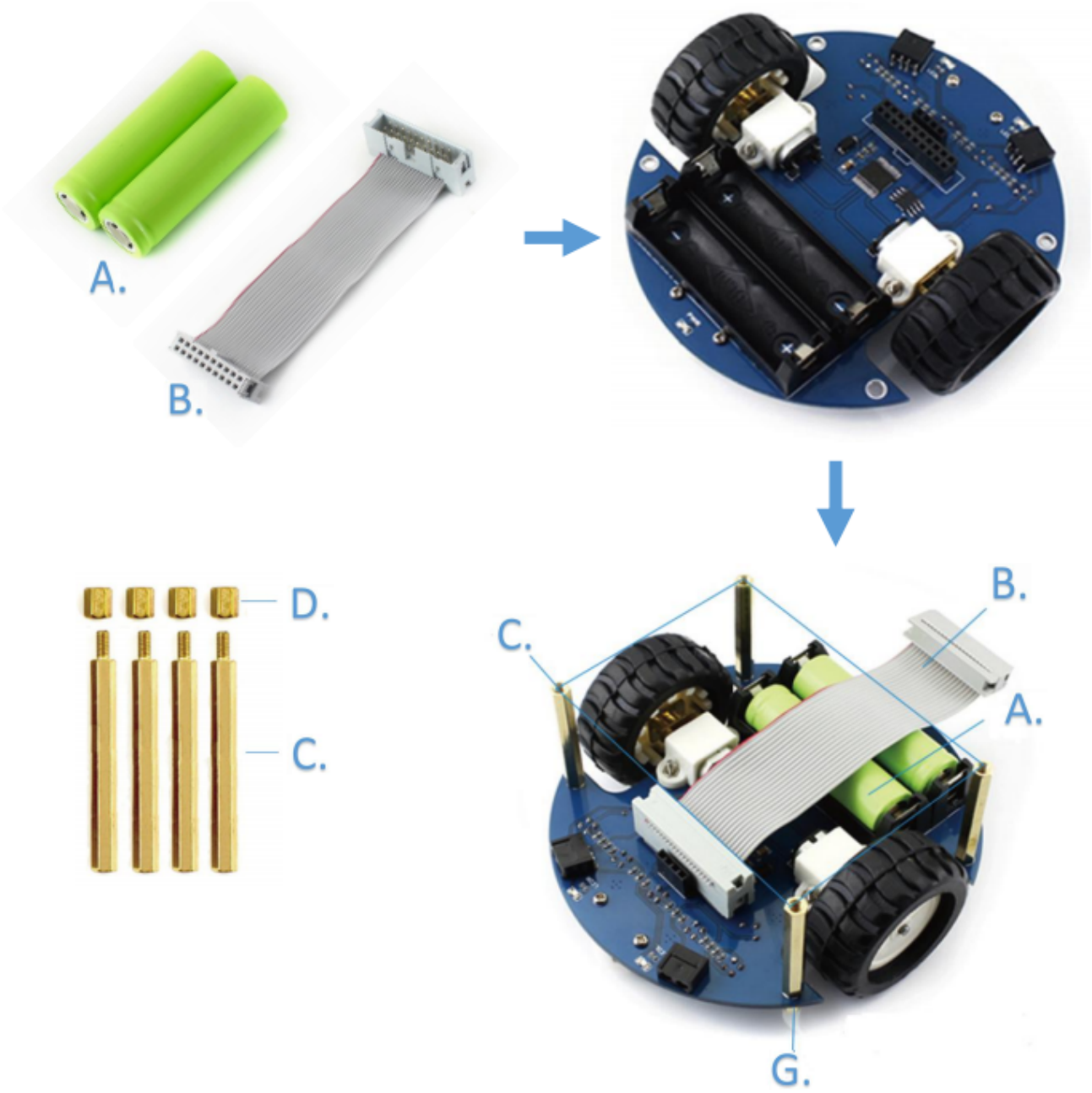
1. Интерфейс управления шасси для подключения к шасси.
2. Интерфейс для подключения Raspberry Pi.
3. Интерфейс подключения сервомоторов.
4. Порт micro USB с USB-UART преобразователем для подключения к Raspberry Pi по UART.
5. LM2596 DC-DC преобразователь.
6. TLC1543 - 10-битный аналогово0цифровой преобразователь для подключения аналоговых датчиков к Raspberry Pi.
7. PCA9685 - 16-канальный 12-битный ШИМ-модуль для работы с сервомоторами.
8. CP2102 - USB-UART преобразователь.
9. Джойстик.
10. Инфракрасный приемник.
11. Пьезодинамик (buzzer).

Сборка робота

Оригинальная инструкция по сборке тут (https://www.waveshare.com/w/upload/1/1a/Alphabot2-pi-assembly-diagram-en.pdf), а так же в папке docs (см. Alphabot2-pi-assembly-diagram-en.pdf).

Сборка шасси робота

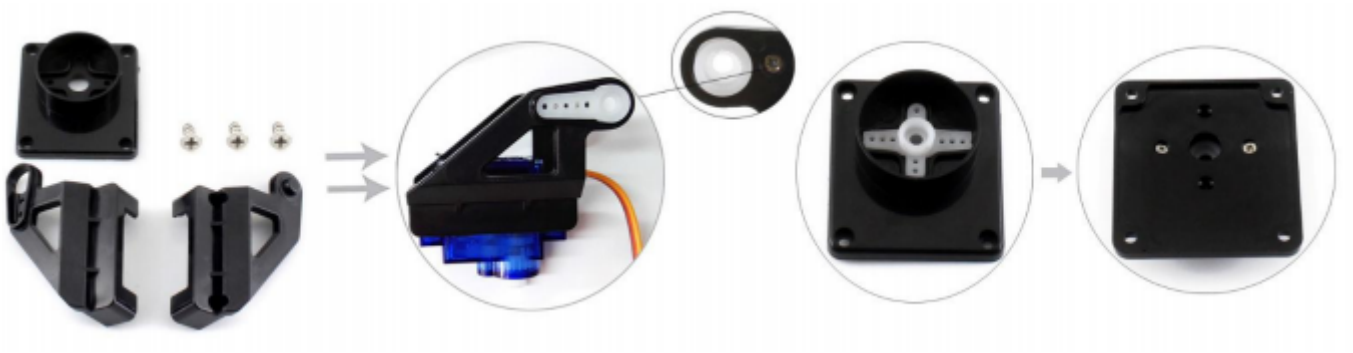
Установите две батареи 14500 **(A)** на базу-шасси AlphaBot2, восьмисантиметровый кабель FC-20P **(B)** вставьте в соответствующий разъем шасси. Вкрутите стойки **(C&D)**.



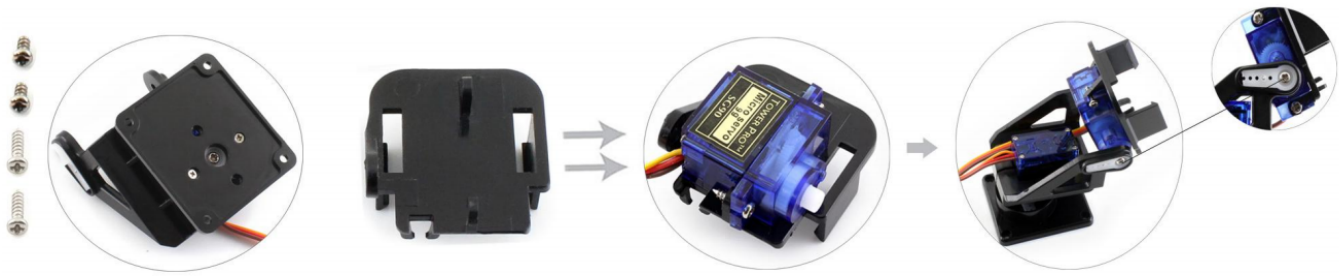
Сборка стойки для камеры

Поместите сервопривод между стойками **(f)** и **(g)** и скрепите винтами (7). Обратите внимание на то,чтобы ротор сервопривода был установлен в правильном направлении.

Установите насадку **(c)** на сервопривод в специальную выемку в корпусе. Если она не подходит **(g)** под выемку в корпусе - слегка подрежете ее. Закрепите с помощью винтов **(5)**. Далее установите крестообразную насадку **(b)** в основание стойки. Также при необходимости подрежете. Закрутите винты **(5)** с задней стороны основания стойки.



Вставьте в крестообразную насадку сервопривод и закрепите винтом **(1)**. Второй, еще никуда не установленный, сервопривод прикрепите двумя винтами **(7)** к подставке для камеры **(e)** как показано на рисунке ниже. Установите подставку под камеру **(e)** и сервопривод на собранную ранее стойку **(f&g)** и прикрутите насадку к сервоприводу с помощью винте **(1)**.

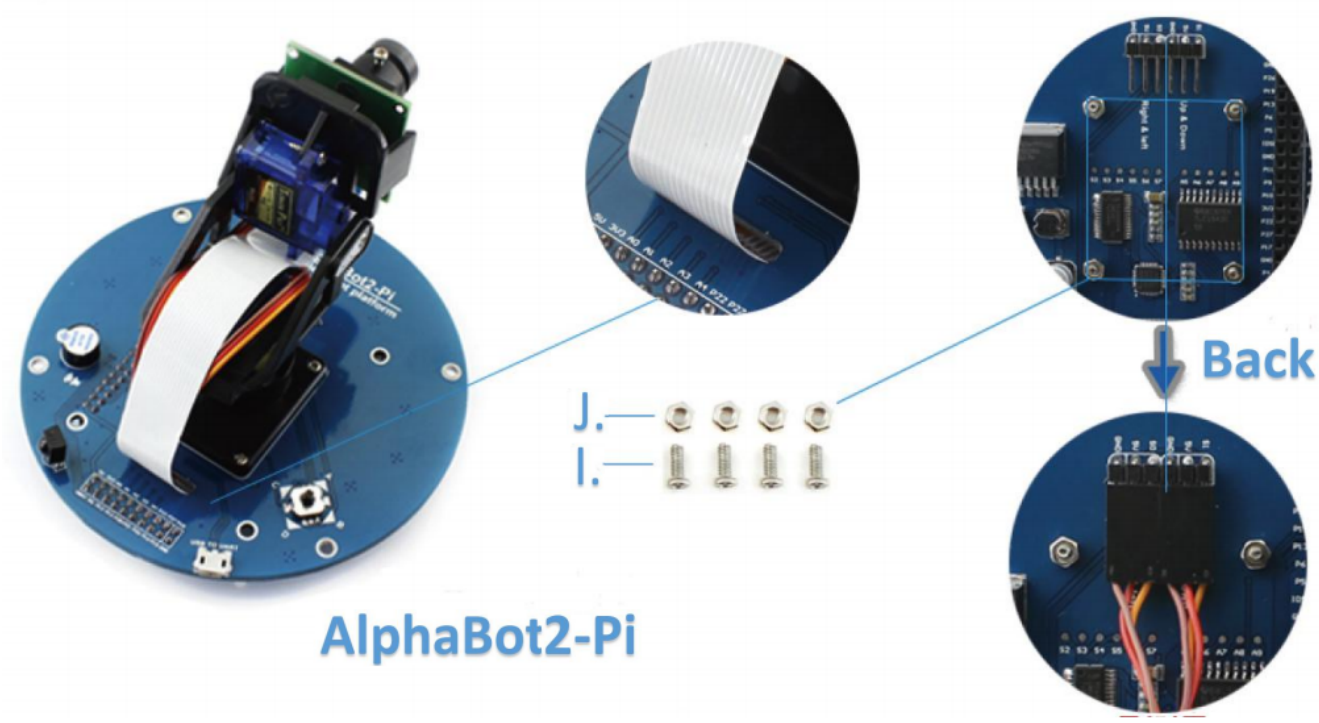


Вставьте двадцатипяти пиновый FFC кабель в соответствующий разъем на модуле камеры для Raspberry Pi и установите модуль в подставку **(e)**.

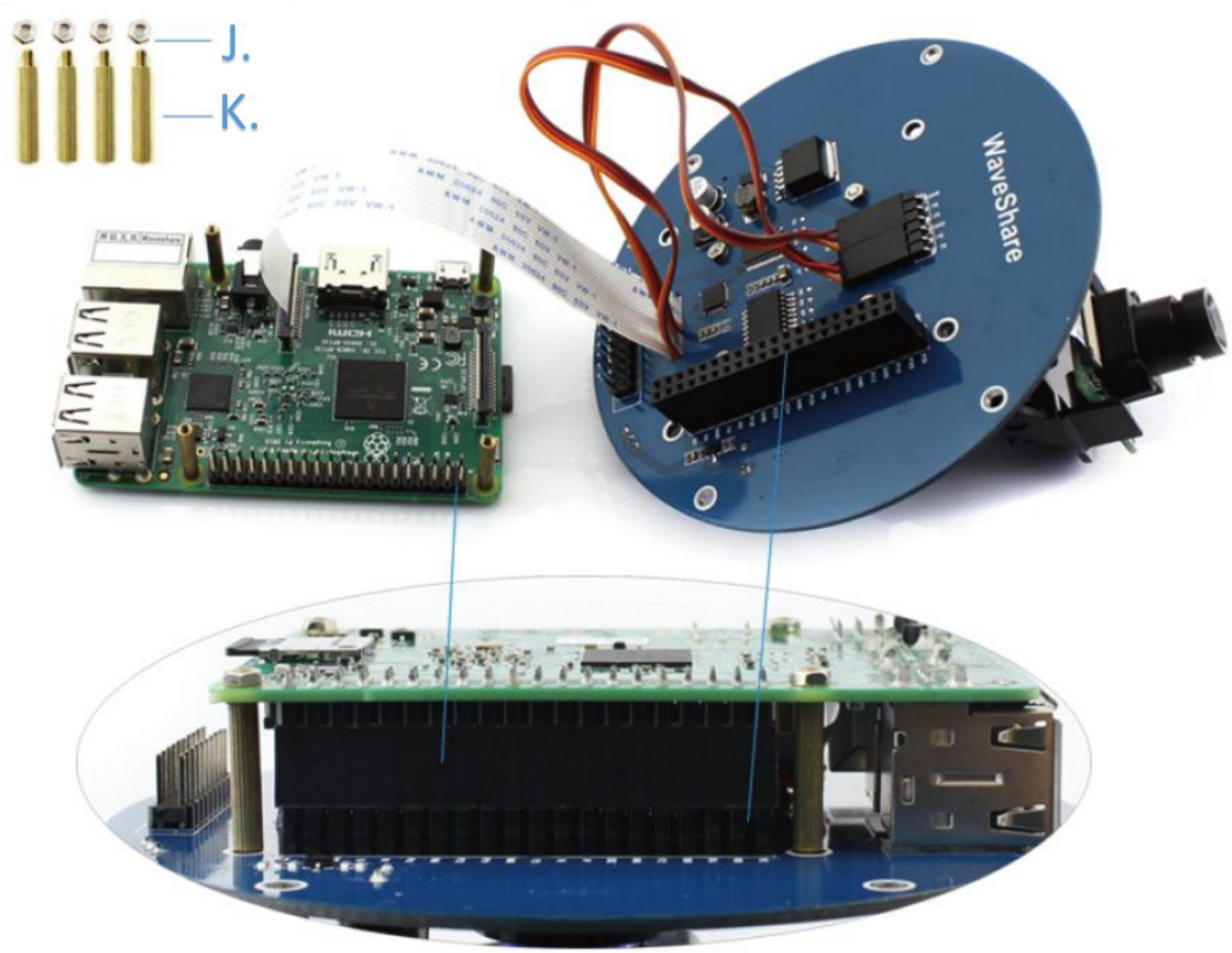


Финальная сборка робота

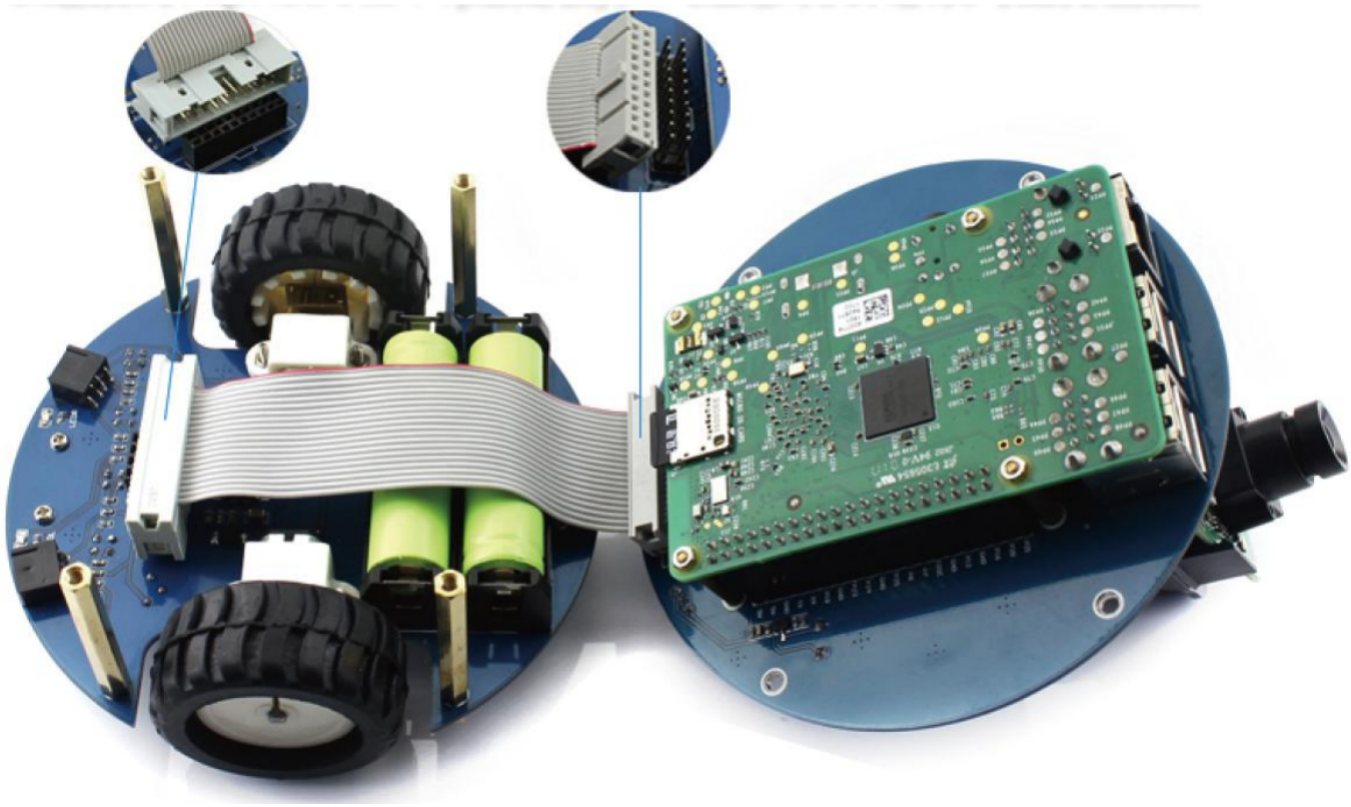
Закрепите стойку для камеры на плате-адаптере с помощью винтов и гаек **(I&J)**, как показано на рисунке ниже. Проведите кабели от сервоприводов через отверстие в нижней части платы-адаптера и вставьте коннектор в соответствующие пины на обратной стороне платы-адаптера. Через то же отверстие проведите шлейф от камеры и вставьте его в соответствующий разъем на Raspberry Pi.



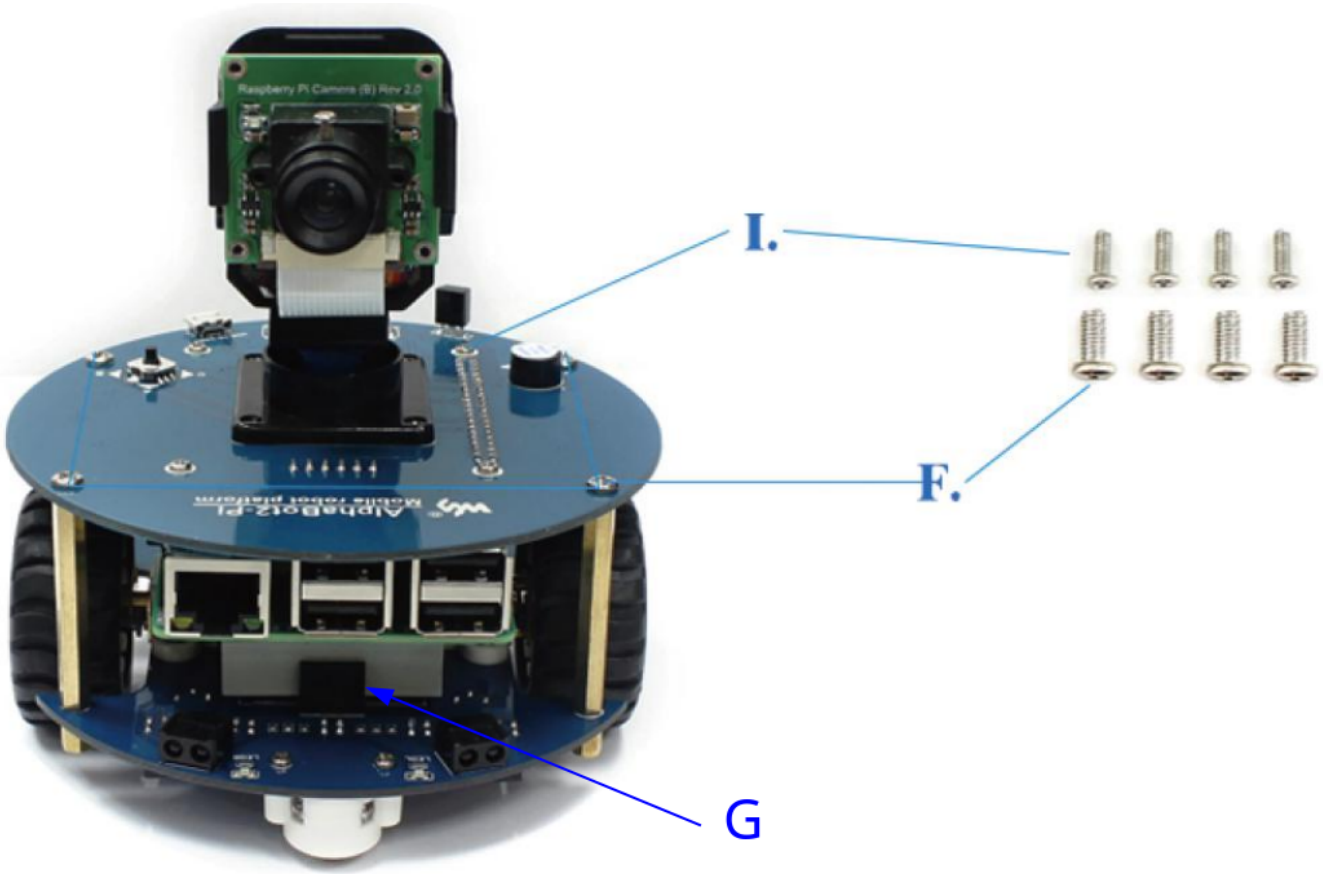
Соедините Raspberry Pi и плату-адаптер с помощью стоек для печатных плат **(I&J)**.



Вставьте кабель FC-20P, установленный на шасси робота, в плату адаптер. Обратите внимание, что коннектор должен вставляться в разъем в соответствие с шелкографией на печатной плате!



Установите ультразвуковой дальномер в соответствующий разъем на шасси робота **(G)**. Соедините плату-адаптер с Raspberry Pi с помощью винтов **(I)**. Соедините шасси робота и плату-адаптер с помощью винтов **(F)**.



ОВНИМАНИЕ! РОБОТ МОЖНО СТАВИТЬ НА ЗАРЯДКУ ТОЛЬКО КОГДА ОН ВЫКЛЮЧЕН! ВО ВРЕМЯ ЗАРЯДКИ ТАКЖЕ НЕЛЬЗЯ ПОДКЛЮЧАТЬСЯ К RASPBERRY PI!

Установка системы и минимальная настройка

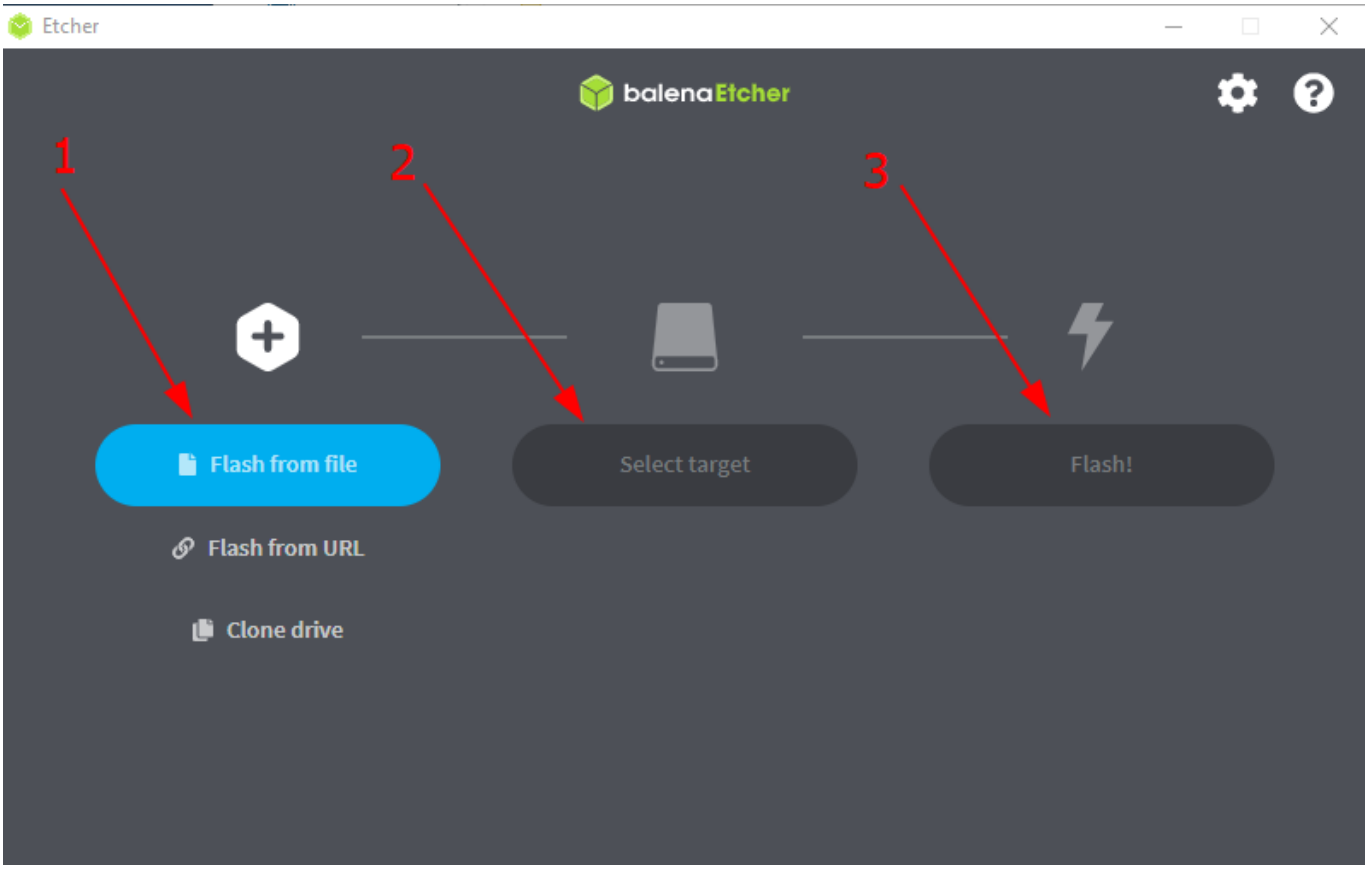
В этом разделе мы рассмотрим процесс установки операционной системы на Raspberry Pi и рассмотрим способы взаимодействия с ней в операционной системе linux и windows. Предполагается что изучающие это руководство владеют linux консолью, и языком программирования python. Для того чтобы запустить linux на Raspberry Pi необходимо выполнить следующие действия:

1. Скачать отсюда (https://www.raspberrypi.org/softwanumpy>=1.20.0re/operating-systems/#raspberry-pi-os-32-bit) образ `Raspberry Pi OS` . Я рекомендую использовать `Raspberry Pi OS with desktop and recommended software`, этот образ будет содержать большинство вещей необходимых нам в будущем.

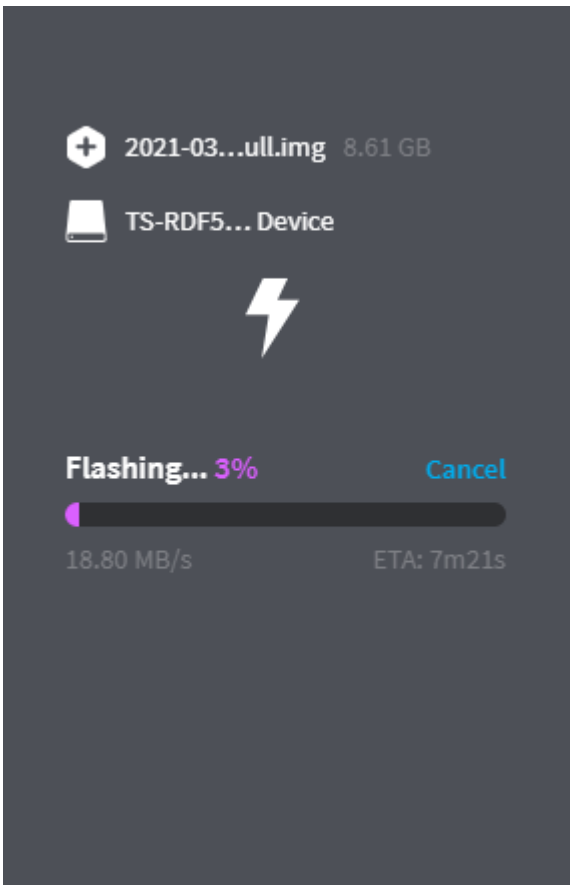
2. Распаковать архив, в Windows можно использовать программу Winrar.

3. Записать образ на карту памяти *(обратите внимание, что объем памяти должен быть больше 4Gb)*. Создатели Raspberry Pi советуют использовать для этого программу Etcher (https://www.balena.io/etcher/). Скачиваем ее и устанавливаем себе на компьютер. Нас встретит простой интерфейс. Сначала нужно выбрать образ, потом выбрать карту память на которую мы хотим его записать, и после согласится на запись. Все ваши данные будут удалены с данной

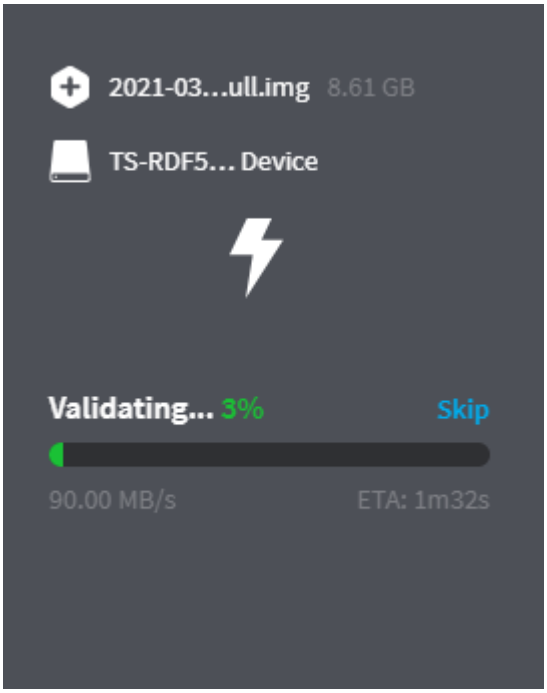
карты. Цифрой 1 на изображении указано, где выбирать образ, 2 - где выбирать карту памяти, 3 - начало прошивки.

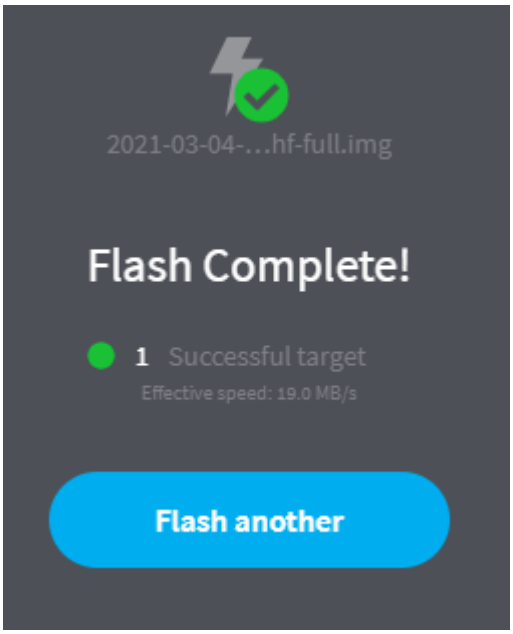


Далее необходимо подождать пока не закончится загрузка на карту памяти.



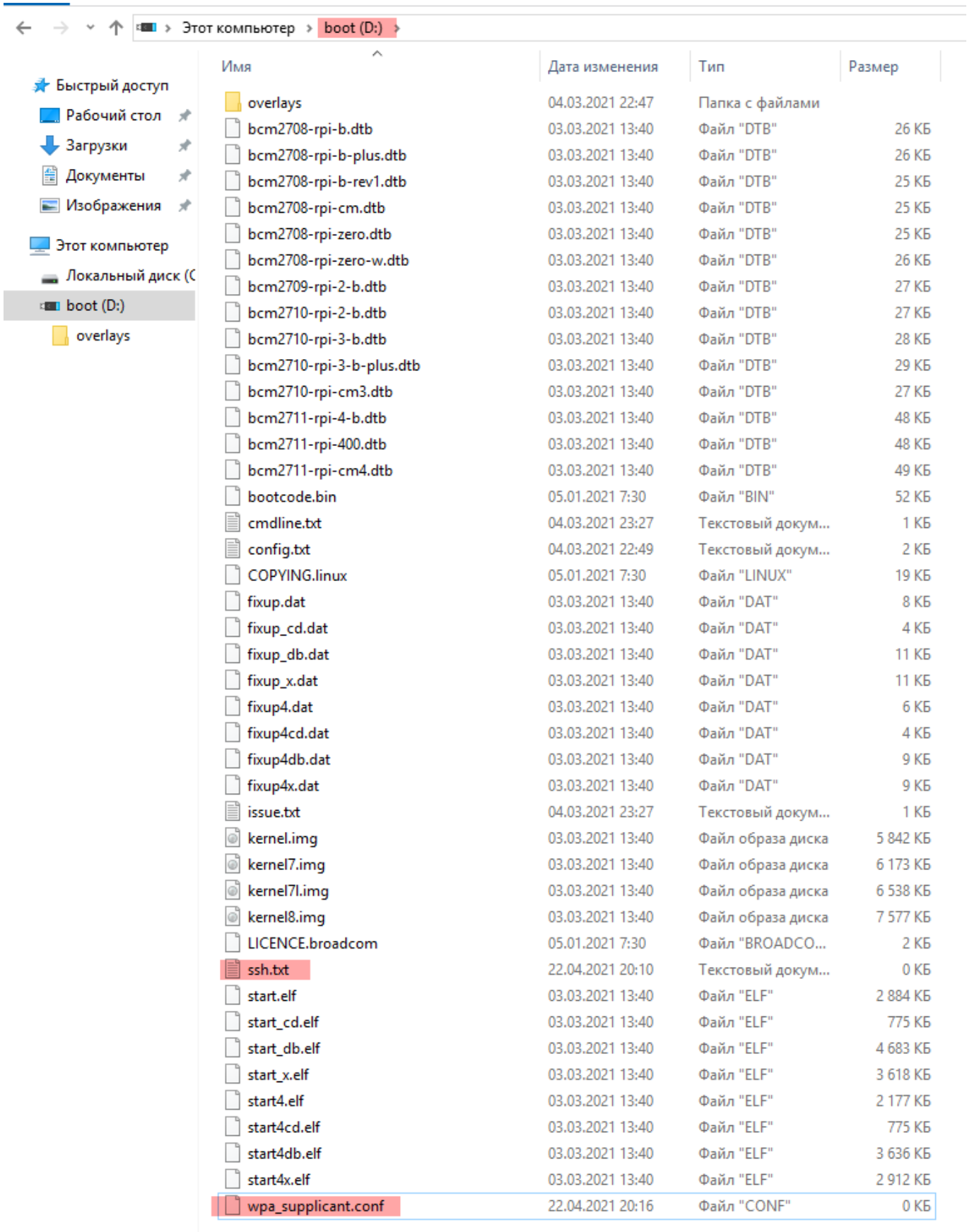
После чего необходимо дождаться завершения проверки правильности записанных данных на флеш карту.





4. Для того, чтобы избавиться от необходимости работать с Raspberry Pi, подключая к ней монитор, клавиатуру и мышку, мы будем использовать подключение с удаленного компьютера по ssh. Для этого нам нужно будет прописать некоторые настройки Wi-Fi.

На карте памяти, на которую был записан образ операционной системы, теперь появилась папка boot.

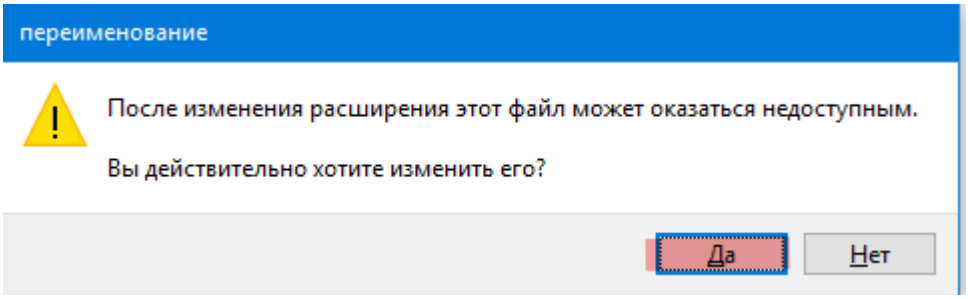


Туда нужно добавить 2 файла:

- ssh.txt - писать в него ничего не нужно, он необходим для того чтобы мы могли подключится к малинке используя сервис ssh (https://ru.wikipedia.org/wiki/SSH)
- wpa_supplicant.conf - тут необходимо приписать параметры вашей Wi-Fi сети в следующем формате:

```
ctrl_interface=DIR=/var/run/wpa_supplicant GROUP=netdev
update_config=1

network={
    ssid="НАЗВАНИЕ ТОЧКИ ДОСТУПА WiFi"
    psk="ПАРОЛЬ WiFi"
    key_mgmt=WPA-PSK
}
```

При создании данного файла система спросит вас:

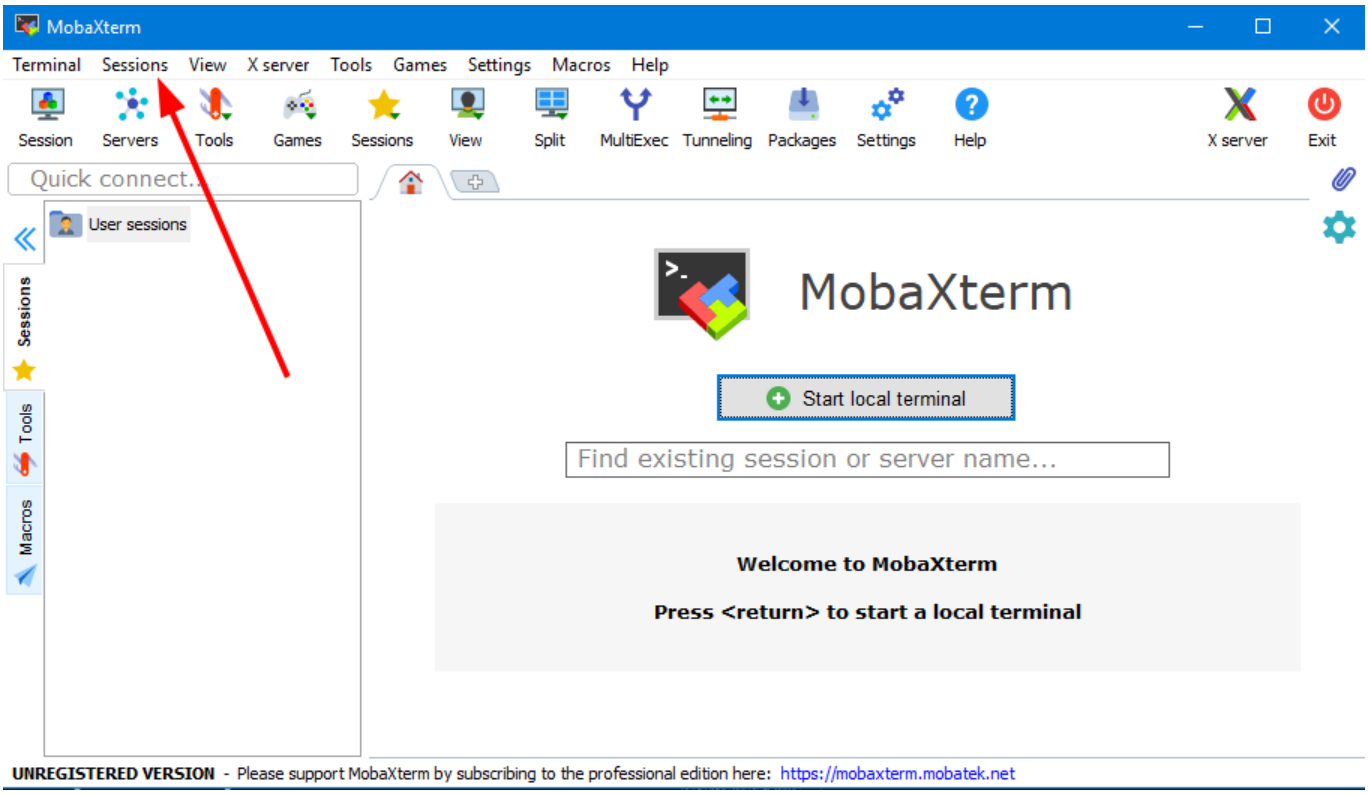


Соглашайтесь. Нажмите правой кнопкой мышки на файл `wpa_supplicant.conf`, выберите `Открыть с помощью`, и откройте с помощью блокнота. Добавьте в открывшийся документ строки, указанные выше, заменив название точки доступа и пароль на свои. Сохраните. Содержимое этого файла будет автоматически перенесено в `/etc/wpa_supplicant/wpa_supplicant.conf` при старте системы.

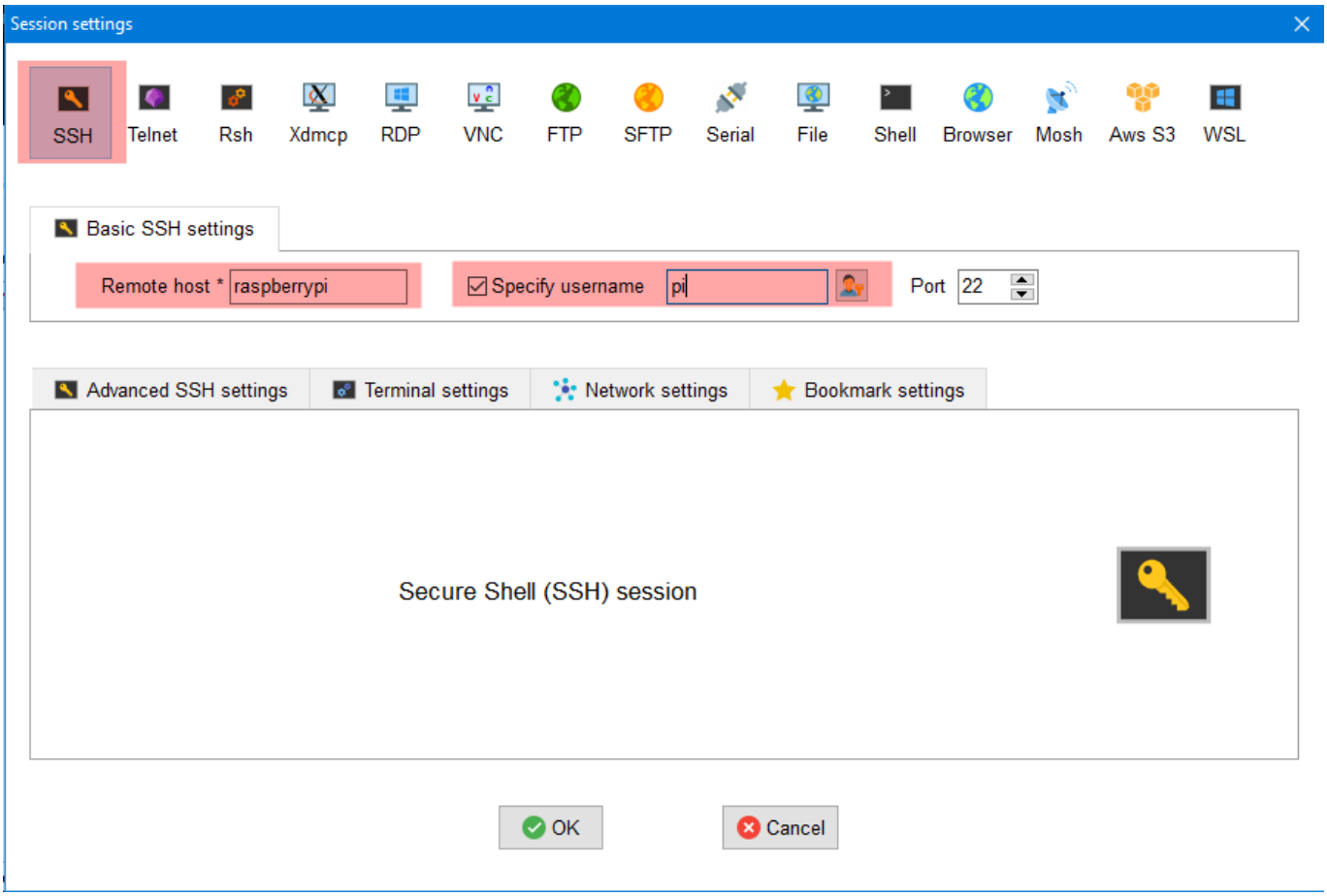
5. Запуск робота. Далее извлекаем карту памяти из компьютера, предварительно воспользовавшись функцией безопасного извлечения, после чего вставляем карту в разъем на Rasbperry Pi, установленную в роботе. `ВНИМАНИЕ! Для того, чтобы на этом этапе не возникло проблем, убедитесь в том, что аккумуляторы заряжены. Это очень важно, так как отключение питания во время загрузки системы может привести к неправильной настройке системы. Во время работы с Raspberry Pi робот не должен заряжаться!` Переводим `PWR SWITCH` находящийся снизу робота из состояния `OFF` в `ON` для включения робота, после чего ждем около 3х минут чтобы система успела применить все настройки. Для того, чтобы на этом этапе не возникло проблем убедитесь что аккумуляторы заряжены, это очень важно потому что робот в неактивном состоянии когда находится на зарядке.

6. Перейдем к подключению к Raspberry Pi. Для этого воспользуемся бесплатной версией программы MobaXterm, которая позволит нам совершить подключение gj практически любому протоколу. (Скачать можно здесь: https://mobaxterm.mobatek.net/). Установите программу на свой компьютер. Для подключения к Raspberry Pi по ssh сделайте следующее:

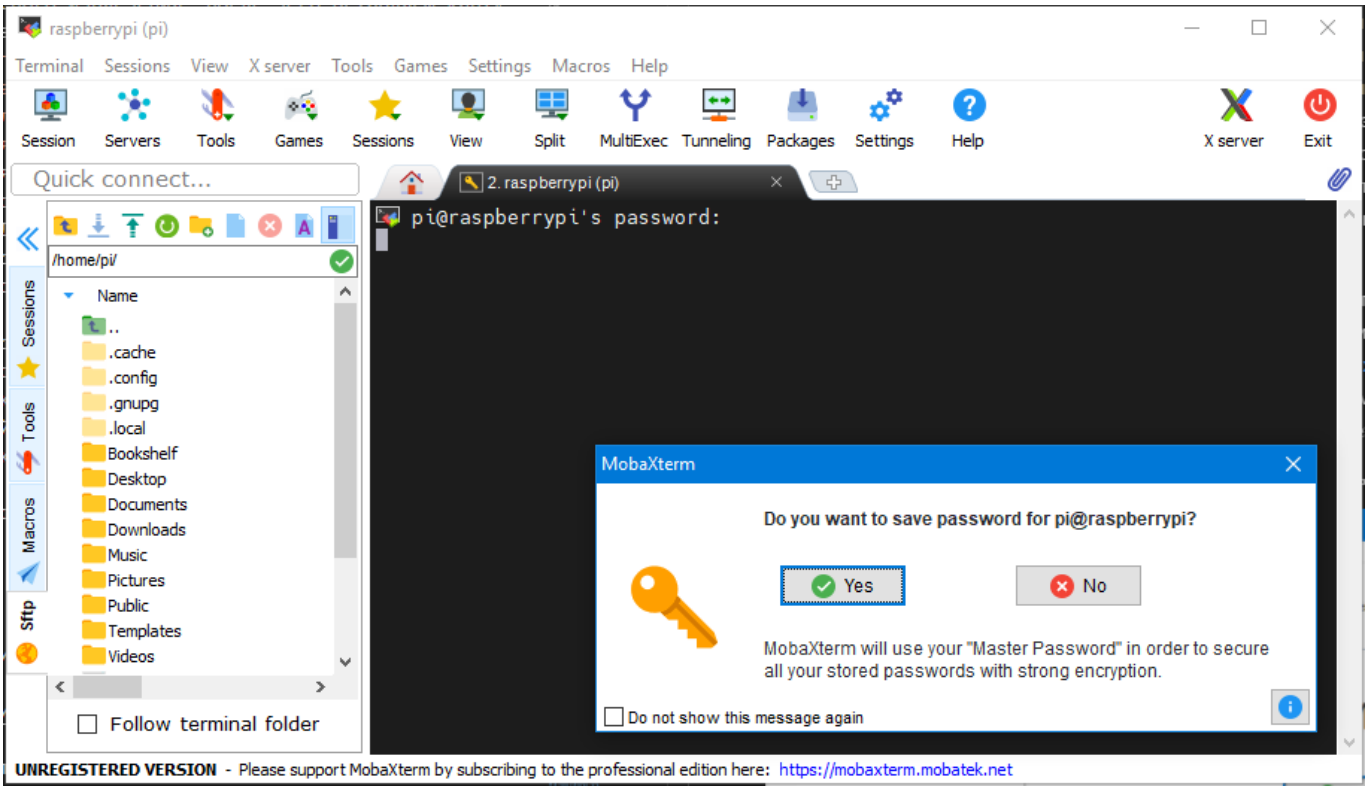
Нажмите на кнопку Session в левом верхнем углу программы, выберите пункт `New session`:



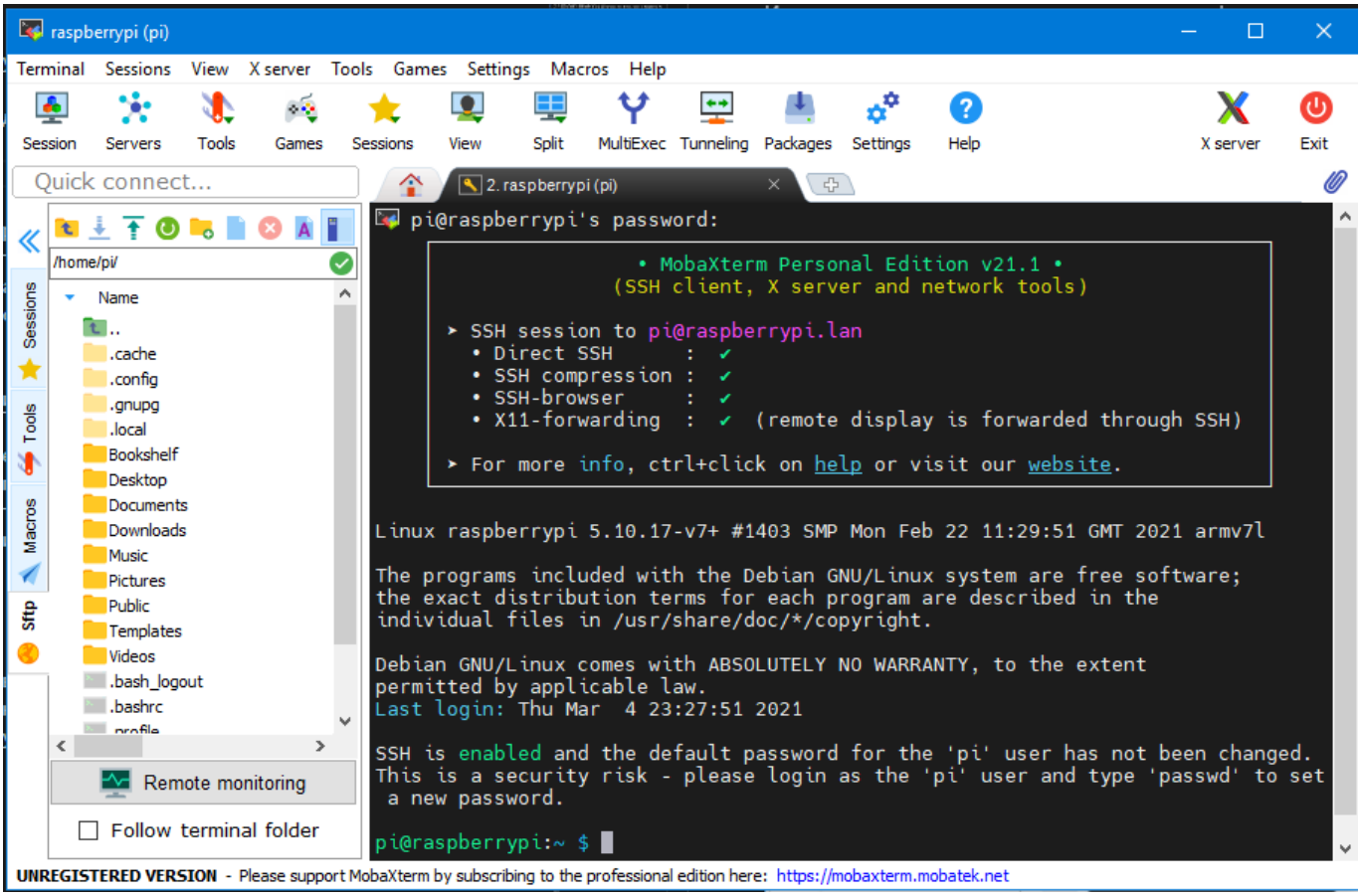
Далее в появившимся окне установите параметры в соответствии с изображением:



Иными словами вы должны подключится по ssh к хосту с названием raspberrypi и именем пользователя pi. Далее вам предложат ввести пароль:



Стандартный пароль для Raspberry Pi OS - `raspberry`, введите его. После чего вы можете сохранить данный пароль в системе, если вам будет так удобнее. После чего система предложит вам ввести команды на исполнение:



Если вы видите такой же вывод, что и на изображении выше - значит все заработало. Если нет, это может свидетельствовать о проблемах с сетью. Первым делом подключитесь к роутеру (обычно он

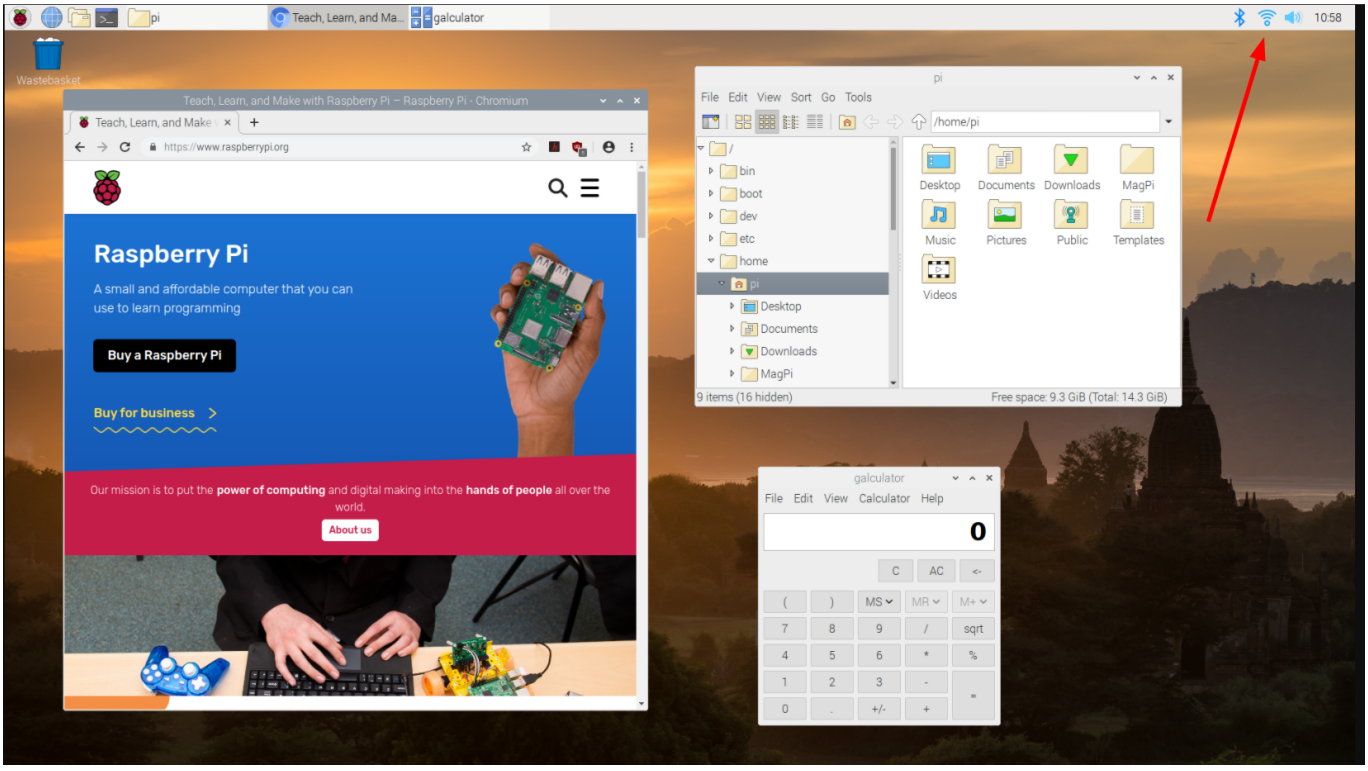
находится на адресе http://192.168.0.1/ или http://192.168.1.1/) и посмотрите список подключенных устройств.

Если это произошло значит все хорошо и все работает если нет, это свидетельствует о проблемах с сетью. Первое что можно сделать это подключится к роутеру (обычно он находится на адресе http://192.168.0.1/или http://192.168.1.1/) и посмотреть там в списках клиентов Raspberry Pi. На моем роутере это выглядит так:

Active DHCP Leases

Hostname	IPv4-Address
raspberrypi	192.168.2.108
android-1718d679845c07df	192.168.2.181
Computer	192.168.2.232
-	192.168.2.204
AW17R3	192.168.2.248

Оттуда можно узнать IP адрес. Далее повторите действия пункта 6, только вместо хоста с названием Raspberry Pi используя полученный IP. Если это тоже не работает остается только одно - отсоединить Raspberry Pi от робота, подключить к монитору и с помощью клавиатуры и мыши через графический интерфейс подключить Raspberry Pi к сети Wi-Fi.



7. Для работы с камерой вам потребуется доступ к рабочему столу Raspberry Pi, для этого нужно установить необходимые пакеты, чтобы иметь возможность видеть рабочий стол.

> Для дальнейшей работы с этим сервисом нужно выполнить настройки из раздела `Настройка робота`.

Для начала установим на Raspberry Pi программы, необходимые для трансляции рабочего стола:

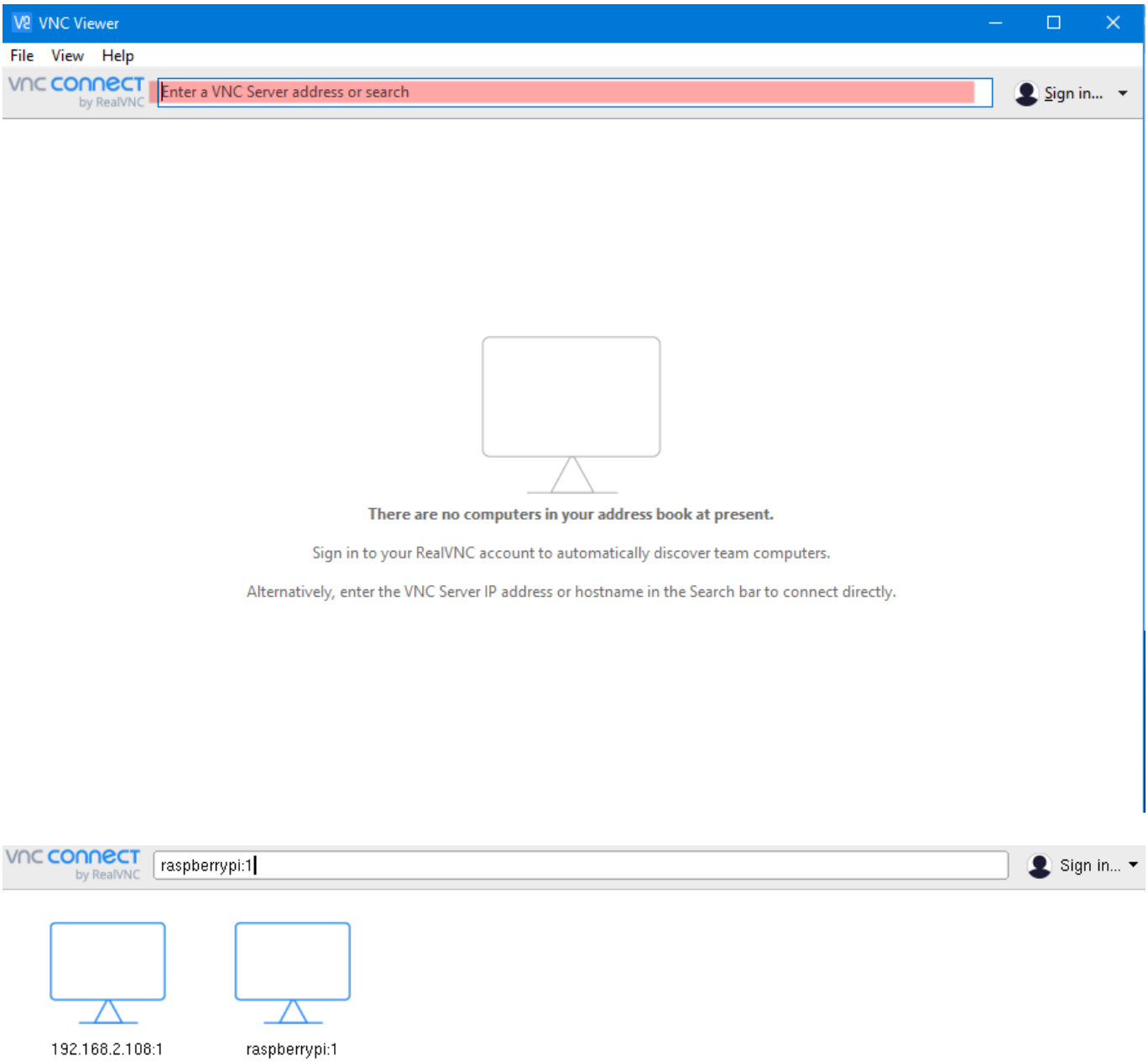
```
sudo apt install realvnc-vnc-server realvnc-vnc-viewer
vncserver
```

После исполнения последний команды вы увидите схожий вывод в терминале:

```
Running applications in /etc/vnc/xstartup

VNC Server catchphrase: "Shampoo canoe Kevin. Little exit druid."
             signature: e9-55-a0-df-85-12-3f-d2

Log file is /home/pi/.vnc/raspberrypi:2.log
New desktop is raspberrypi:1 (192.168.2.108:1)
```

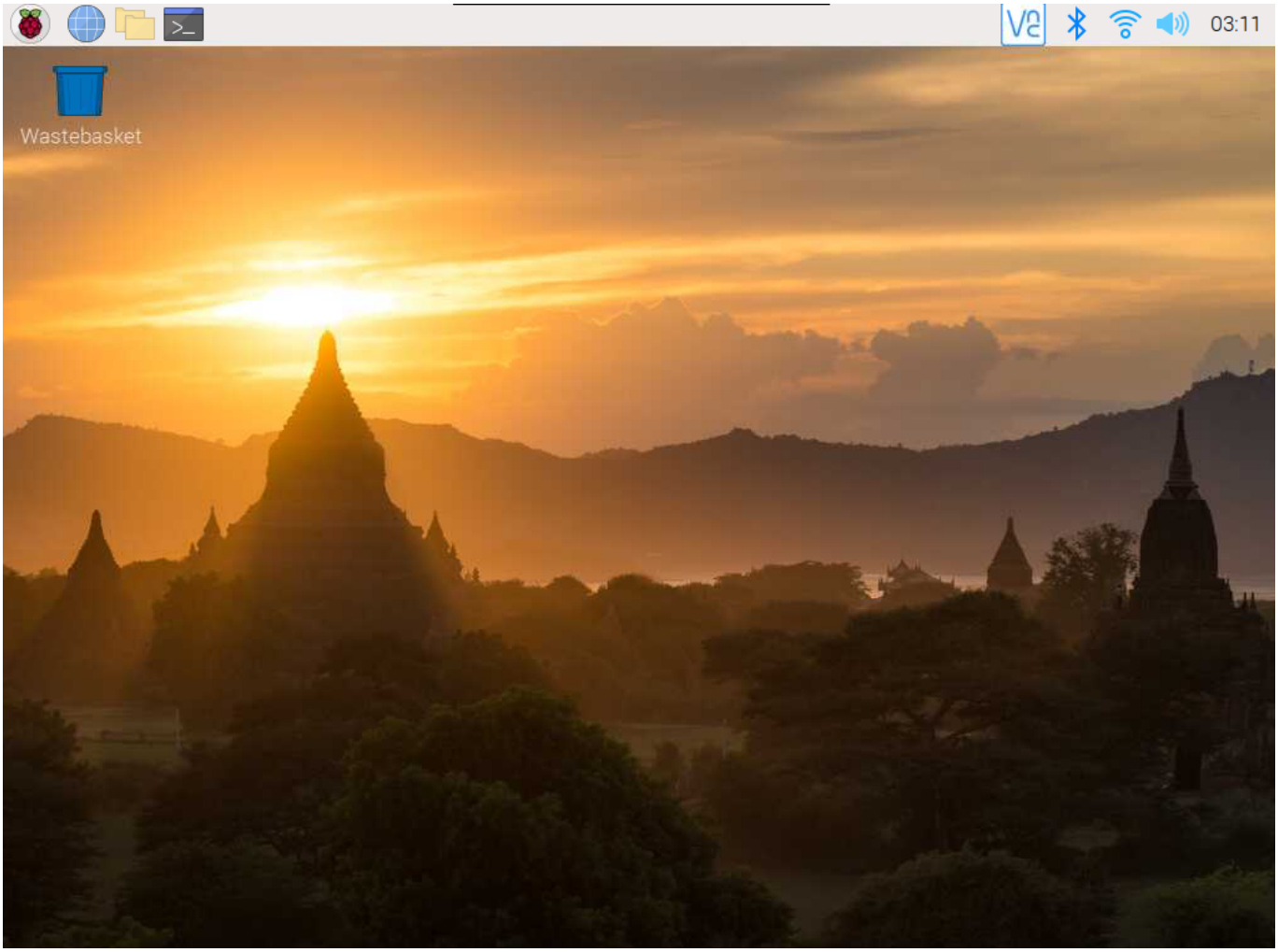
Нужно запомнить `raspberrypi:1`, эта строчка будет использоваться как адрес в клиенте.

Теперь переходим к вашему рабочему компьютеру - откройте ссылку (https://www.realvnc.com/en/connect/download/viewer/), и скачайте версию программы, подходящую под вашу операционную систему. Установите и запустите программу. После включение вы увидете окно, вверху которого расположена форма для ввода. Введите туда `raspberrypi:1` и нажмите Enter.


VNC Viewer
File View Help
vnc connect by RealVNC
Enter a VNC Server address or search
Sign in...
There are no computers in your address book at present.
Sign in to your RealVNC account to automatically discover team computers.
Alternatively, enter the VNC Server IP address or hostname in the Search bar to connect directly.

vnc connect by RealVNC
raspberrypi:1
Sign in...
192.168.2.108:1
raspberrypi:1

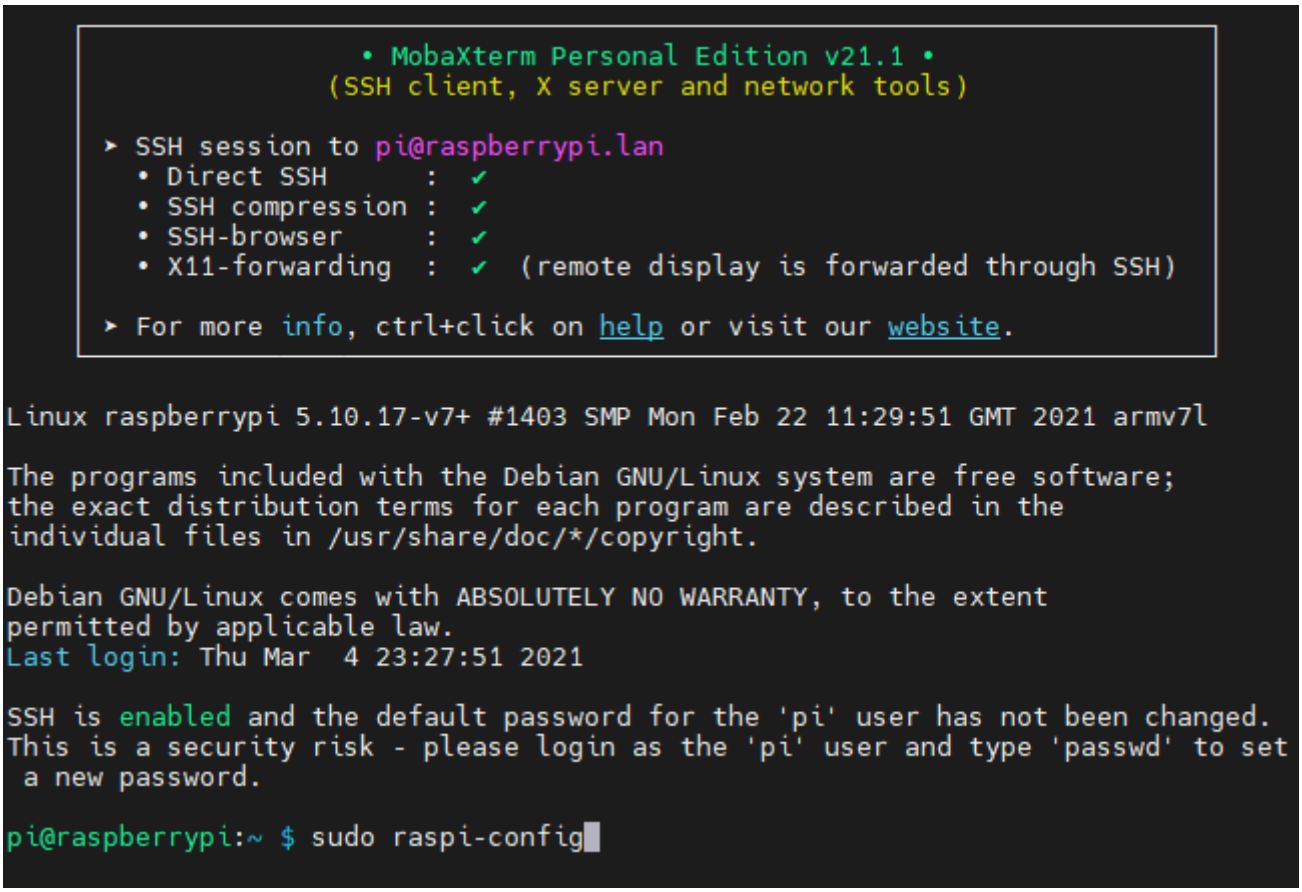
Вам предложат ввести логин и пароль от Raspberry Pi. Они стандартные - логин это `pi`, а пароль - `raspberry`. После этих действий вы увидите рабочий стол Raspberry Pi.



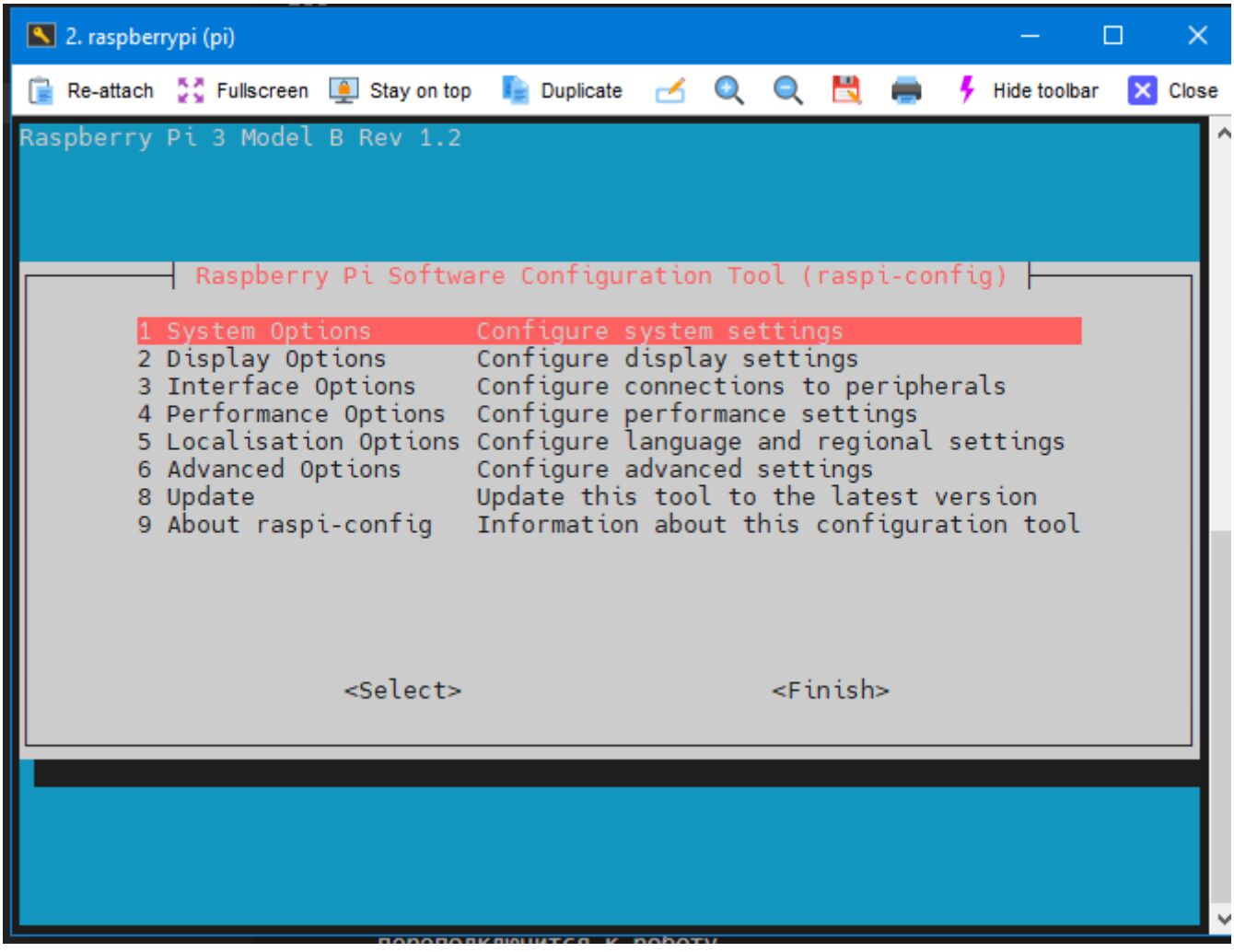
Настройка робота.

Для работы с роботом необходимо выполнить ряд настроек. Для начала их нужно будет сделать в специальной консольной программе для конфигурирования Raspberry Pi - `raspi-config`, Для чего используем следующую команду:

```
sudo raspi-config
```



После чего вы увидене следующее меню:



Если вам будет удобнее, терминал можно открепить от программы ModaXterm и перенести в любую область рабочего стола - просто нажмите левой кнопкой мыши на вкладку с терминалом и перетащите в свободную область рабочего стола.

Навигация в данном режиме очень проста. Стрелками вверх и вниз мы выбираем нужный пункт меню, enter позволяет пройти в подпункты выбранного пункта или выбрать конкретную настройку. Стрелки вправо и влево позволяют выбрать один из пунктов, находящихся внизу (Select и Finish на изображении выше). Ниже перечислено, какие настройки необходимо сделать:

```
Interface Options -> Camera -> Yes -> OK
Interface Options -> SPI -> Yes -> OK
Interface Options -> I2C -> Yes -> OK
Interface Options -> Serial Port -> No -> Yes -> OK
Interface Options -> VNC -> YES
```

После завершения настройки выберите finish и система предложит вам перезагрузить Raspberry Pi, на что соглашаемся. Подключится можно будет способом описанным выше, либо нажать кнопку R и MobaXterm автоматически постарается переподключиться к роботу.

После того как система загрузится установите пакеты, необходимые для работы с периферией робота. Для этого сначала обновляем систему, потом ставим все необходимое:

```
sudo apt update && sudo apt dist-upgrade -y
sudo apt install ttf-wqy-zenhei python-pip python-smbus python-serial
sudo pip3 install RPi.GPIO spidev rpi_ws281x
```

Исполнение первой команды может занять достаточно длительное время.

Упровление компонентами робота

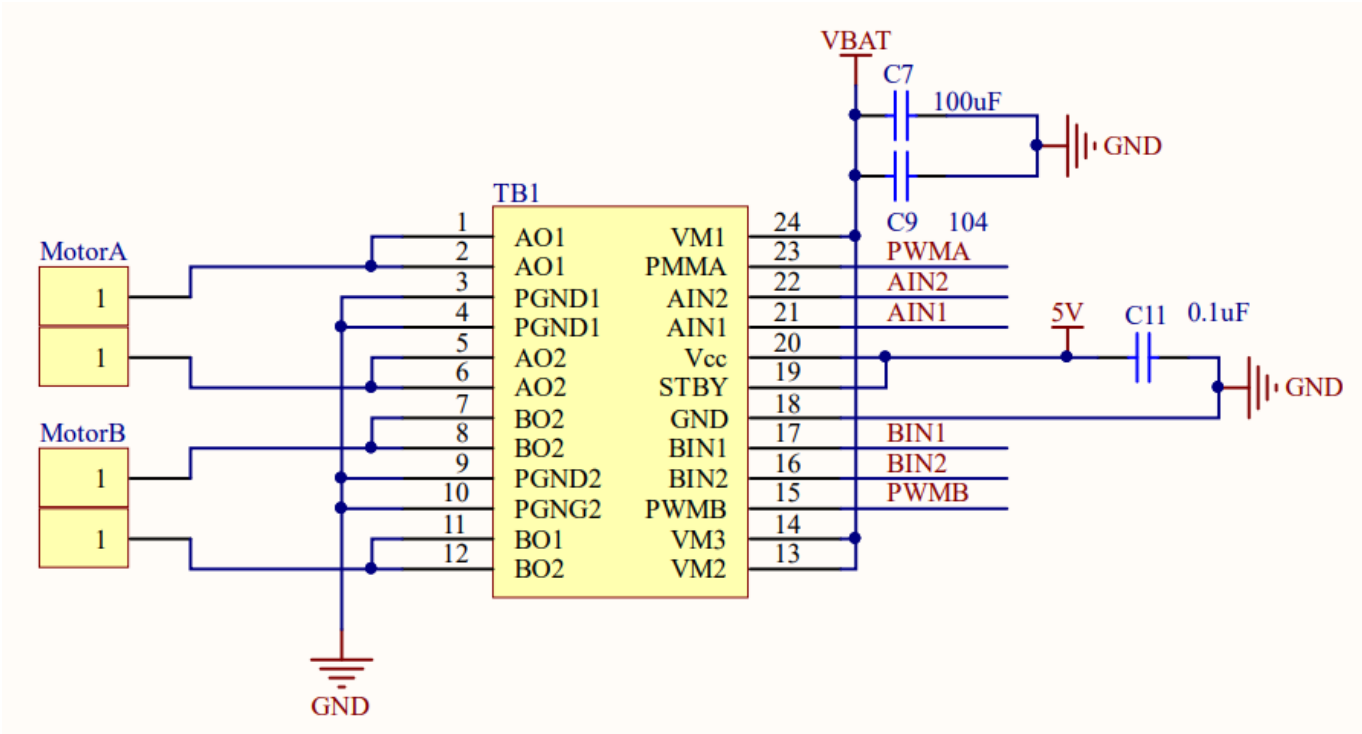
Для управления роботом предлагается использовать библиотеку bsp.py. Для того, чтобы воспользоваться ей, ее нужно поместить в папку с вашим проектом и добавить ее в ваш код следующим образом:

```
from bsp import *
```

Рассмотрим все компоненты робота и то, как ими можно управлять, какие данные с них можно получить.

Управление моторами

Рассмотрим управление моторами в роботе. Для этого на плате-шасси установлен двухканальный мостовой драйвер (2 H-моста) моторов TB6612FNG. На изображении ниже вы видите часть принципиальной схемы платы-шасси, отвечающей за управление моторами:

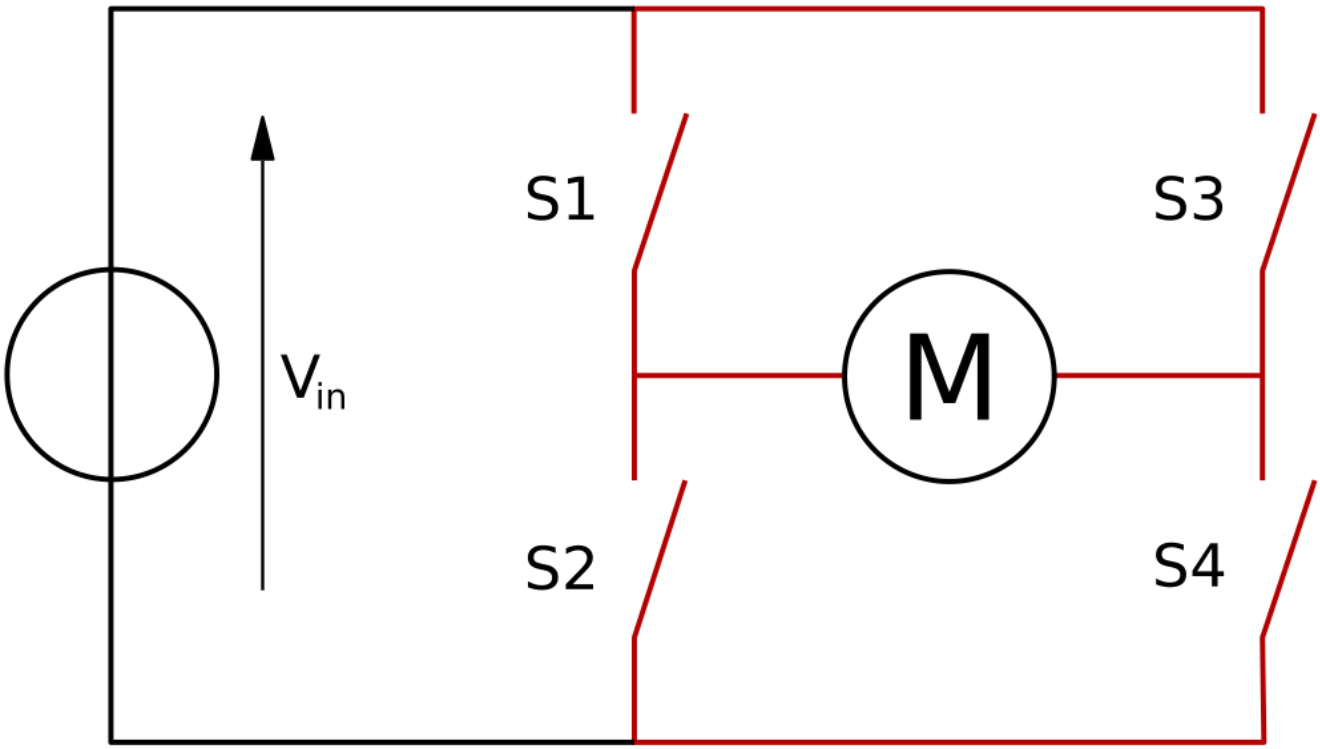


Рассмотрим назначение выходов микросхемы:

- PWMA\PWMB - вход для управления скоростью вращения мотора, для канала A и B (соответственно, для первого и второго мотора) с помощью ШИМ (*англ. PWM*);
- AIN1\AIN2 - входы полумостов канала A;
- BIN1\BIN2 - входы полумостов канала B;
- A01\A02 - выходы полумостов канала A;
- B01\B02 - выходы полумостов канала B;
- STBY - включение микросхемы;
- VM - вход питания силовой части микросхемы, двигателей;
- VCC - вход питания логической части микросхемы;
- GND - земля.

H-мост

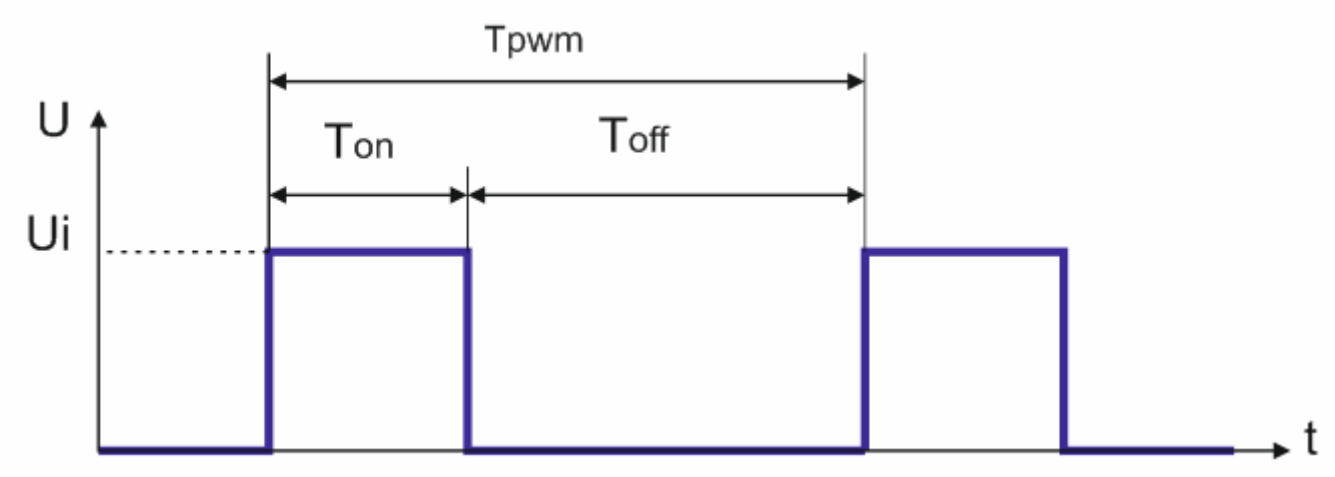
H-мост (англ. H-bridge) - это схема, позволяющая приложить напряжение к нагрузке в разных направлениях.



H-мост состоит из четырех переключателей, которыми могут быть, например, транзисторы или реле. Когда ключи S1 и S4 замкнуты, а S2 и S3 разомкнуты, мотор крутится в одну сторону, если же замкнуть S2 и S3, а S4 и S1 разомкнуть - мотор будет крутиться в другую сторону.

ШИМ

ШИМ (широтно-импульсная модуляция, англ. PWM) - способ управления мощностью на нагрузке с помощью изменения скважности импульсов при постоянной амплитуде и частоте импульсов.



Основные параметры ШИМ-сигнала:

- Ui - амплитуда импульсов;
- Ton - время, когда сигнал включен;
- Toff - время, когда сигнал отключен;
- Tpwm - время периода ШИМ;

Мощность на нагрузке пропорциональна времени включенного и отключенного сигнала.

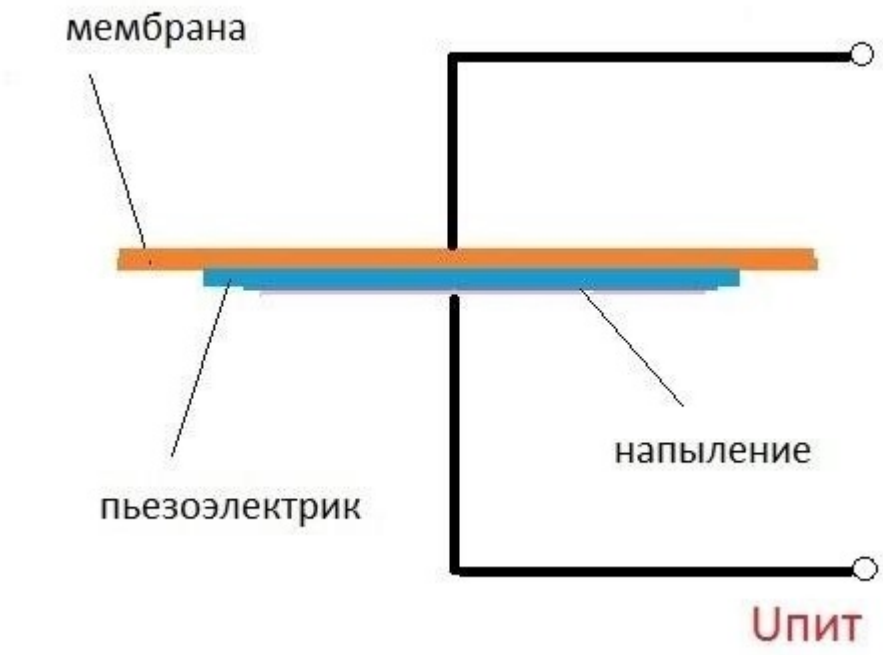
Вернемся к части принципиальной схемы, отвечающей за управление моторам. Видно, что оно осуществляется с помощью четырех пинов GPIO, которые задают направление вращения двух моторов (AIN1\AIN2, BIN1\BIN2) и двух пинов с функцией ШИМ (PWMA\PWMB), которые задают скорость вращения.

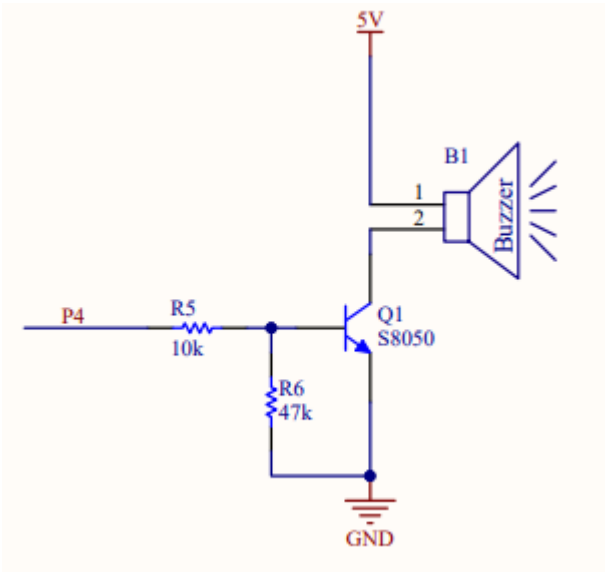
В библиотеке bsp.py реализован функционал, который позволит вам просто объявить объект-мотор, задать скорость и направление вращения:

```
from bsp import *

m = motor()
speed = 10          # Скорость, по сути напрядение на моторе (0 .. 100)
m.forward(speed)    # Ехать вперед со скоростью speed
time.sleep(1)       # Задержка
m.backward(speed)   # Ехать назад со скоростью speed
time.sleep(1)
m.left(speed)       # Крутиться вправо со скоростью speed
time.sleep(1)
m.right(speed)      # Крутиться влево со скоростью speed
time.sleep(1)
m.stop()            # Остановиться

m.setMotor(10, 20)  # Выставить скорость (ШИМ) 10 на левом колесе, и 20 на
правом.
# Скорость может быть от -100 до 100, знак определяет направление
time.sleep(1)
```

Взаимодействие с пьезодинамиком (buzzer)

Пьезодинамик (англ. buzzer) - устройство на основе обратного пьезоэлектрического эффекта, который заключается в механической деформации пьезоэлектрика под воздействием электрического поля. Простыми словами - при подаче напряжения на пьезодинамик, устройство переводит его в колебание мембраны динамика.





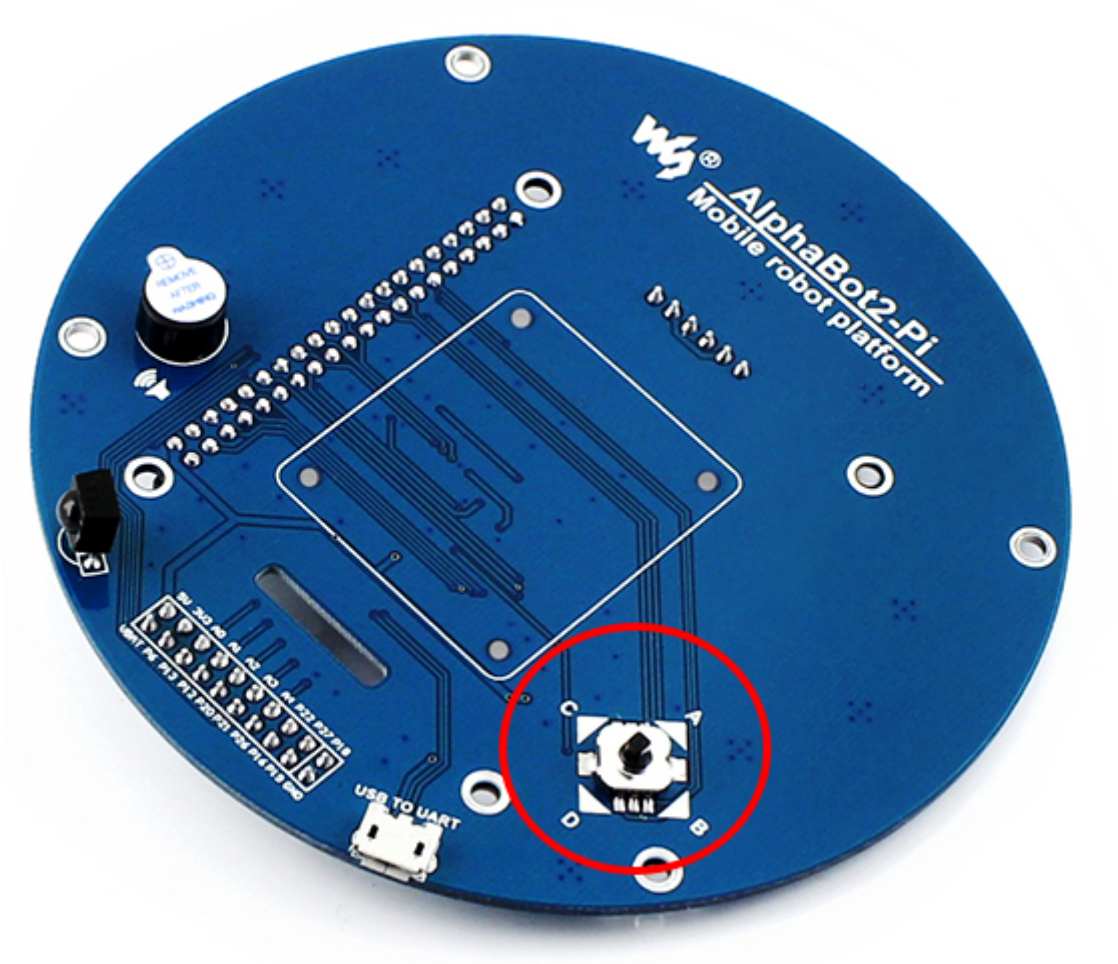
Взаимодействие с пьезодинамиком с помощью библиотеки bsp.py реализовано следующим образом:

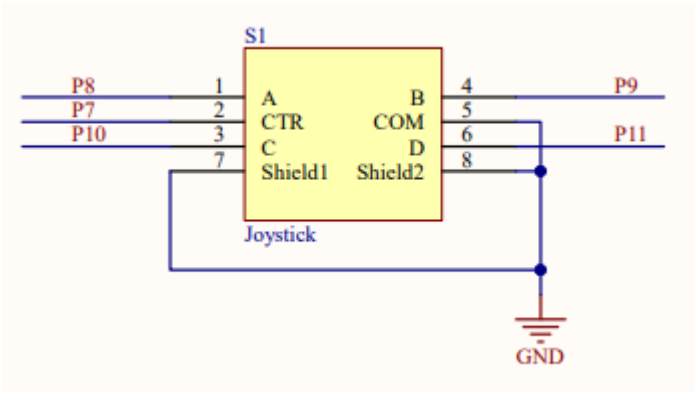
```
from bsp import *

b = beep()
b.on()              # Включить пищалку
time.sleep(1)       # Задержка
b.off()             # Выключить пищалку, пищит до тех пор пока не выключите
time.sleep(1)
```

Джойстик

На верхней плате-адаптере робота расположен джойстик. На изображении он обведен в красный кружок. Джойстик может возвращать информацию об отклонении его влево, вправо, вперед и назад,

а так же о центральном расположении.





Средствами bsp.py вы можете взаимодействовать с джойстиком следующим образом:

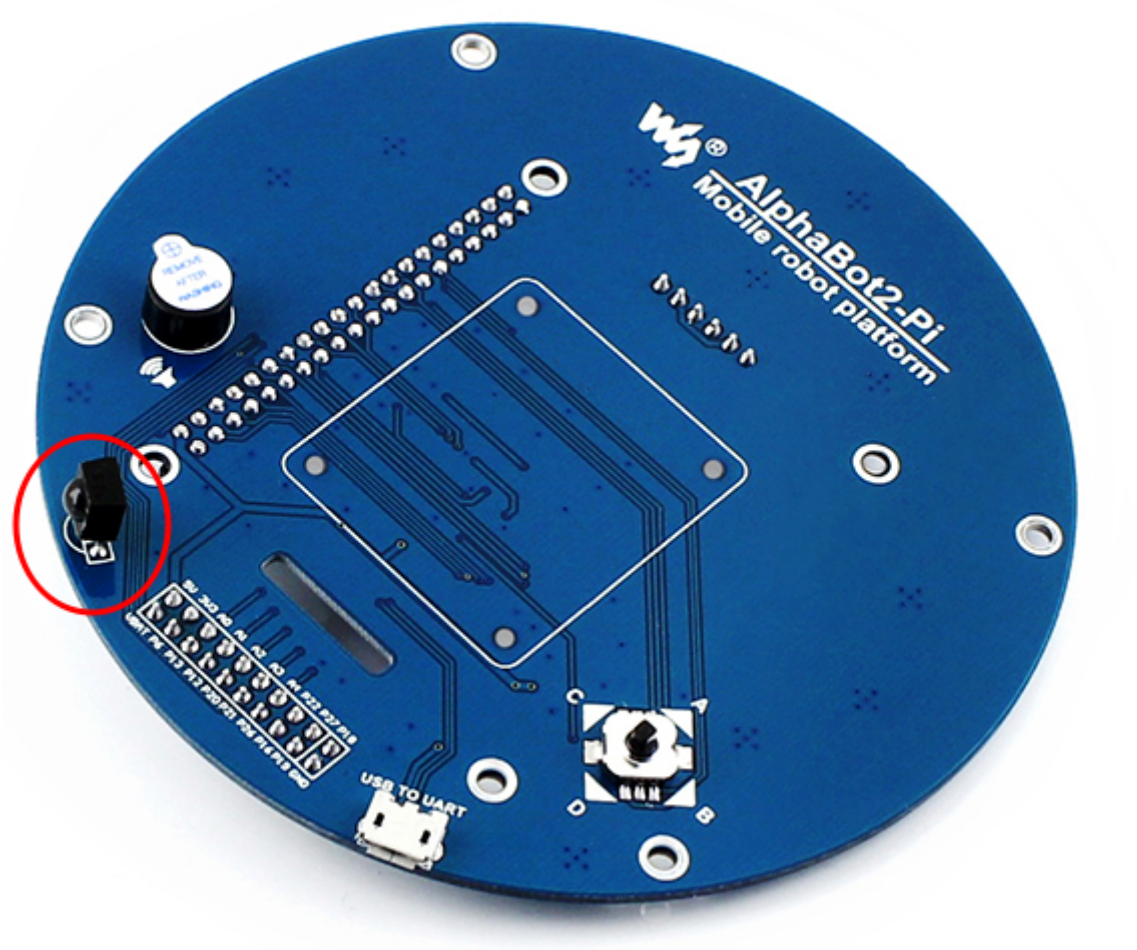
```
j = joystick()
j.check_ctr()       # Вернет единицу если нажата крестовина
j.check_a()         # Вернет единицу если нажата A
j.check_b()         # Вернет единицу если нажата B
j.check_c()         # Вернет единицу если нажата C
j.check_d()         # Вернет единицу если нажата D
```

Рассмотрим пример взаимодействия с джойстиком с помощью bsp.py:

```
from bsp import *

j = joystick()
b = beep()
while True:          # В бесконечном цикле проверяем нажата ли кнопка
    if j.check_ctr():
        b.on()       # Если нажата, то включаем пьезодинамик и печатаем
слово
        print("center")
        while j.check_ctr():
            time.sleep(0.01) # Ждем когда кнопку отпустят
    else:
        b.off()      # Когда кнопку отпустили выключим пьезодинамик
```

Управление с пульта

На плате-адаптере расположен инфракрасный датчик, с помощью которого можно осуществлять управление роботом с помощью пульта дистанционного управления



Средствами bsp.py вы можете взаимодействовать с пультом следующим образом:

```
IR = IR_controll()
key = IR.getkey()   # Присваиваем переменной key номер нажатой кнопки на пульте
# Метод вернет None если не одна кнопка не нажата, или вернет код кнопки если ее нажали в данный момент, и repeat до тех пор пока кнопку не отпустят
```

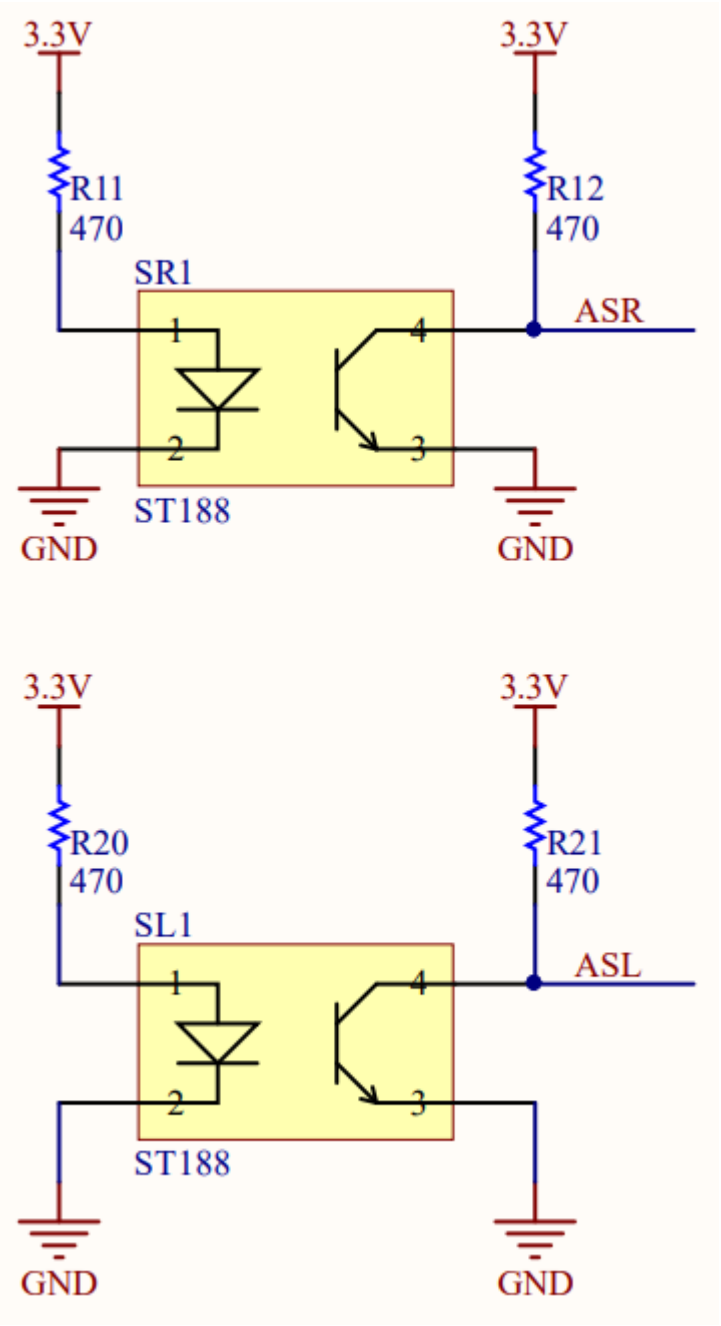
Пример использования - программа которая выводит код нажатой кнопки:

```
from bsp import *

ir = ir_controll()
while True:
```

```
    key = ir.getkey()
    if(key != None and key != "repeat"):
        print(hex(key))
```

Инфракрасные датчики для обнаружения препятствий

На плате-шасси робота расположены два инфракрасных датчика, которые предназначены для обнаружения препятствий роботом.



Нужно понимать, что с помощью такого датчика вы можете получить информацию о факте наличия препятствия, но не о том, как, например, оно далеко от вас. Такому типу датчиков могут помешать засветы, например от солнца.

Средствами bsp.py вы можете взаимодействовать с инфракрасными датчиками следующим образом:

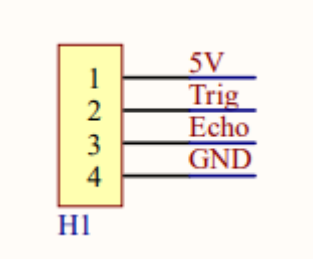
```
b = ir_bumper()
l = b.check_l()     # Вернет 1 если препятствие зафиксировано левым
датчиком
r = b.check_r()     # Вернет 1 если препятствие зафиксировано правым
датчиком
```

Пример кода для работы с инфракрасными датчиками. В консоль выводится информация о сенсорах, которые фиксируют объект:

```
from bsp import *

b = ir_bumper()
while True:
    if b.check_l():
        print ("LEFT")
        time.sleep(0.3)
    if b.check_r():
        print ("RIGHT")
        time.sleep(0.3)
```

Ультразвуковой дальномер

Для обнаружения препятствий и определения расстояний до них можно воспользоваться ультразвуковым дальномером.



Ультразвуковой датчик в формате модуля установлен в специальный разъем на плате-шасси робота.

Ультразвуковой дальномер

Ультразвуковой дальномер генерирует звуковые импульсы и слушает эхо. Замеряя время, за которое отраженная звуковая волна вернется обратно можно определить расстояние до объекта.

В отличие от инфракрасных датчиков, рассмотренных выше, мы можем получить информацию о расстоянии до объекта, так же такому сенсору будут нестрашны засветы от солнца, но он может плохо работать с очень тонкими и пушистыми предметами.

Средствами bsp.py вы можете взаимодействовать с ультразвуковыми датчиками следующим образом:

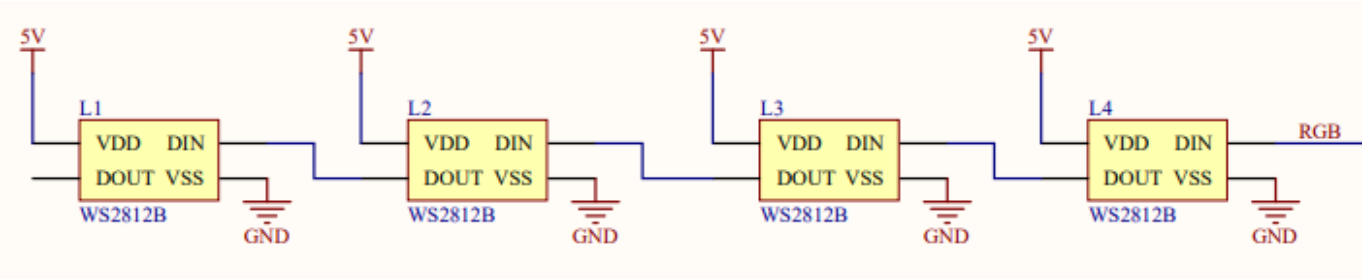
```
u = us_sensor()
k = u.dist()        # Вернет расстояние до объекта, считанное сенсором
```

Пример кода который будет выводить расстояние до объекта до тех пор, пока не будет нажато сочетание клавиш `Ctrl+C`:

```
from bsp import *

u = us_sensor()
try:
    while True:
        print ("Distance:%0.2f cm" %u.dist())
        time.sleep(0.3)
```

```
except KeyboardInterrupt:
    GPIO.cleanup()
```

RGB светодиоды



Для работы с RGB светодиодами необходимо внести некоторые изменения в конфигурацию запуска Raspberry Pi, так как адресные светодиоды WS2812B требуют специфического сигнала управления. Реализовать его средствами linux не возможно, поэтому на помощь нам приходят такие модули как DMA и PWM. DMA позволяет передавать данные на светодиоды минуя центральный процессор, а PWM через управление скважностью позволяет передать данные от DMA в требуемом формате. Но за такую реализацию нужно платить, поэтому одновременно со светодиодами нельзя использовать аудиокарту, встроенную в Raspberry Pi. Для ее отключения нам и нужно исправить настройки запуска следующим образом:

```
sudo nano /boot/config.txt
```

Нужно добавить в конец файла 2 строки:

```
hdmi_force_hotplug=1
hdmi_force_edid_audio=1
```

Также необходимо закомментировать эту строку:

```
dtparam=audio=on
```

таким образом:

```
#dtparam=audio=on
```

Также для работы нужно установить дополнительные библиотеки, для чего сделаем следующее:

```
sudo pip3 install rpi_ws281x adafruit-circuitpython-neopixel
sudo python3 -m pip install --force-reinstall adafruit-blinka
```

> Все взаимодействие с диодами необходимо осуществлять через SUDO, иными словами команда запуска скрипта будет выглядить так:

```
sudo python3 test.py
```

Для примера попробуем установить разные цвета на диоды:

```
from bsp import *

r = rgb_led()
r.color_raw(0, 255, 255, 255)
r.color_raw(1, 0, 0, 255)
r.color_raw(2, 0, 255, 0)
r.color_raw(3, 255, 0, 0)
```

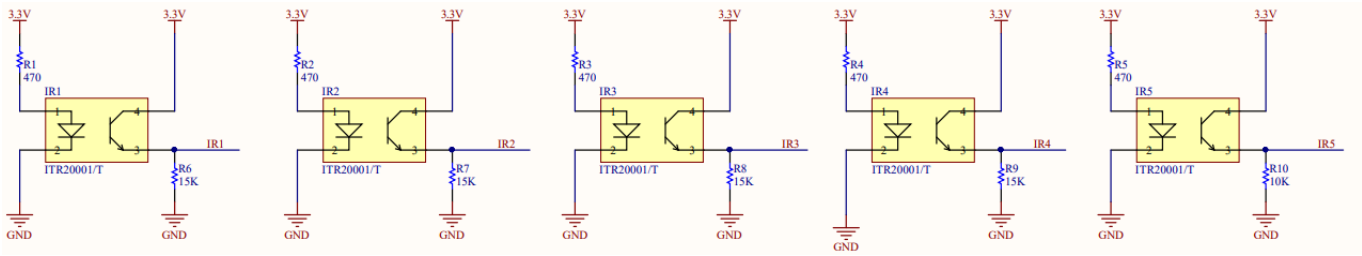
Такого же эффекта можно добиться если указать не яркость каждого отдельного цвета, а применить название соответствующего цвета.

```
from bsp import *

r = rgb_led()
r.color(0, "White")
r.color(1, "Blue")
r.color(2, "Green")
r.color(3, "Red")
```

Доступны следующие цвета:

- Aqua
- Black (выключить диод)
- Blue
- Purple
- Green
- Red
- White
- Yellow

Датчики линии



Для отслеживания линии в нижней части робота есть 5 оптических датчиков. Для примера можно просто считать с них значения:

```
from bsp import *

l = line_sensor()
while True:
    print(l.AnalogRead())
    time.sleep(0.2)
```

Если поставить робота на трассу (ее можно найти в папке docs), то меленькие значения (< 200) будут получены с сенсора который находится над черной линией, а большие (> 800) над белым участком трассы.

Управление положением камеры

Для управлением положением камеры (а оно регулируется с помощью кронштейна с двумя сервоприводами) здесь отвечает специальная микросхема - PCA9685. Принцип работы с ней следующий - ей задается угол поворота камеры и она будет его поддерживать, пока мы не передадим новое значение угла. Для реализации данного функционала можно воспользоваться следующей функцией:

```
c = CameraAngle()
c.setCameraAngle(30, 90)
```

Этот код установит камеру на 30 градусов по вертикальной оси и 90 градусов по горизонтальной.

Пример программы, которая двигает камерой вправо и влево, а также вверх и низ:

```
from bsp import *
c = CameraAngle()

while True:
    for i in range(0, 180, 1):
        c.setCameraAngle(i, i)
        time.sleep(0.02)

    for i in range(180, 0, -1):
```

```
        c.setCameraAngle(i, i)
        time.sleep(0.02)
```

Пример простой программы

Для примера работы с роботом реализуем программу движения по линии с помощью датчиков линии. Данная программа позволит двигаться роботу по замкнутому контуру, который можно получить распечатав страницы 2,2,2,2,3,3,3,3,11 из документа с элементами трассы (документ linefollowtiles.pdf в папке docs).

> Этот пример работает только из под SUDO т.к. в нем используются светодиоды.

```
from bsp import *

m = motor()
r = rgb_led()
l = line_sensor()

colors = {0:4, 1:3, 2:1, 3:0}

delta_sensor = 200  # Пороговое значение срабатывания сенсора

while True:
    c = l.AnalogRead()  # Получаем текущее значение датчика линии

    i = 0
    for i in range(4):
        if c[colors[i]] < delta_sensor:
            r.color(i, "Red")       # Включаем красный на светодиоде,
соотвсетствующем датчику, обнаружевшему черную линиб
        else:
            r.color(i, "Black")     # Выключаем светодиод, если нет
        i = i+1

    # Начинаем поворачивать в зависимости от того, какие датчики заметил
черную линию
    if c[1] < delta_sensor:
        m.setMotor(40,0)
    elif c[3] < delta_sensor:
        m.setMotor(30,10)
    elif c[0] < delta_sensor:
        m.setMotor(10,30)
    elif c[4] < delta_sensor:
        m.setMotor(0,40)
    else:
        m.setMotor(10, 10)
```

В данном примере робот ездит по черной линии, светодиодами подсвечивается сенсор под которым обнаружена линия. delta_sensor - пороговое значения срабатывания сенсора, оно может меняться в зависимости от освещения. Это значение можно узнать если воспользоваться примером, который считывает значения с сенсоров.

Компьютерное зрение

В рамках данного блока предлагается рассмотреть пример, который позволит распознать зеленый круг, а также взаимодействовать с ним. Этот процесс состоит из двух этапов:

1. подбор параметров;
2. непосредственная работа с программой по распознаванию.

Перейдем к примеру:

1. Поставим пакеты для работы с компьютерным зрением:

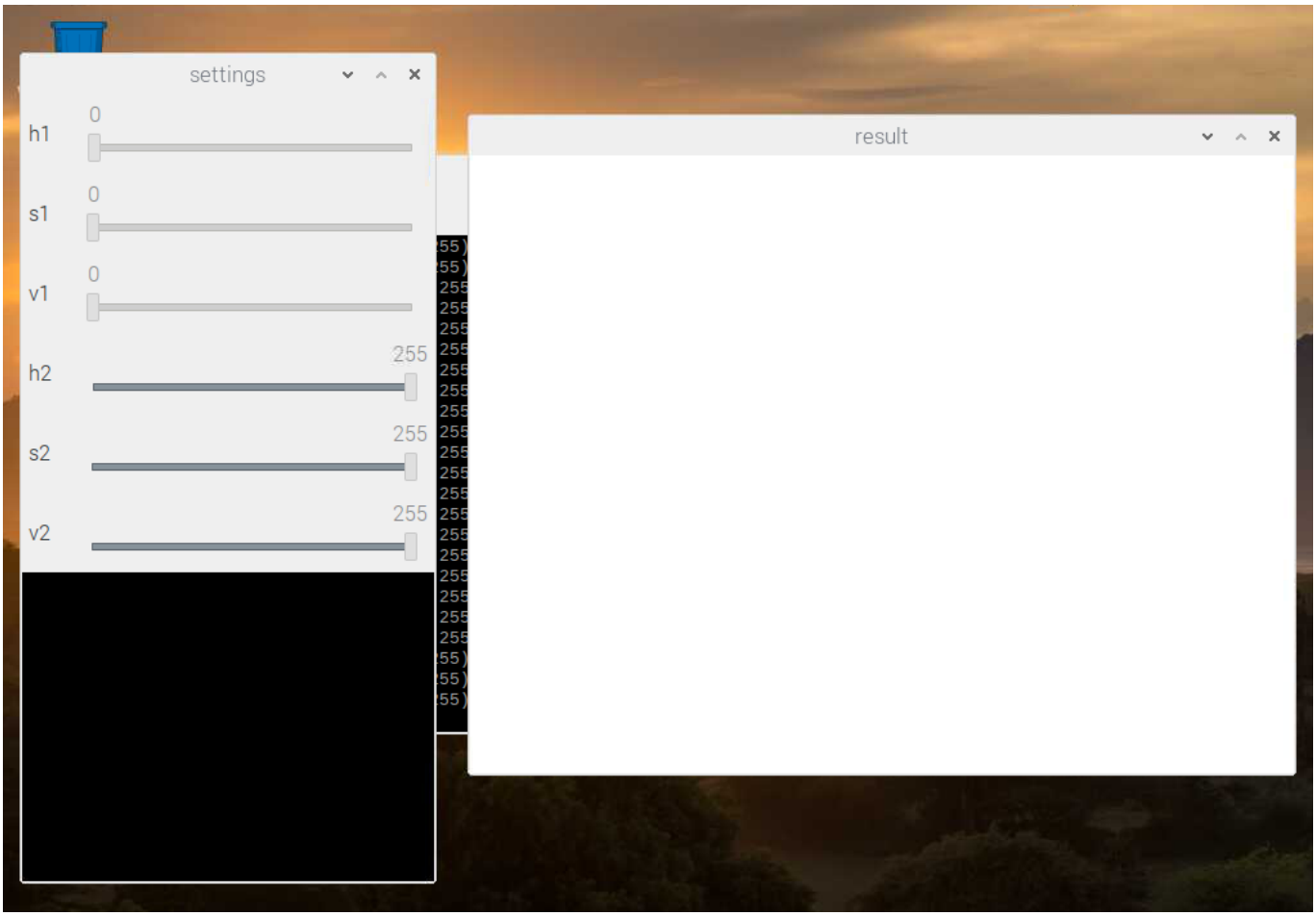
```
sudo apt install libatlas-base-dev
pip3 install numpy opencv-python
```

2. Далее запускаем скрипт, предназначенный для подбора параметров.

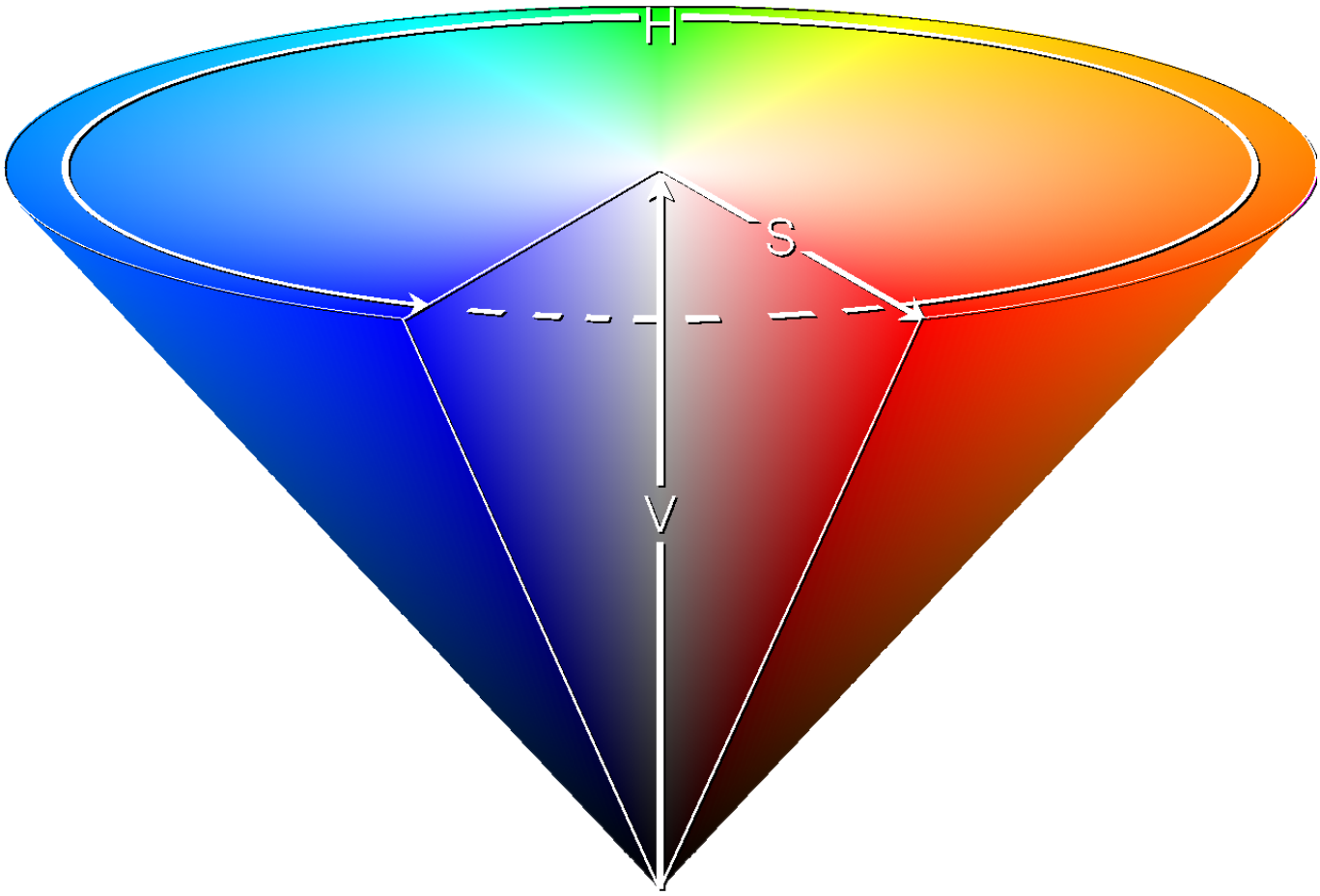
```
import cv2
import numpy as np

if __name__ == '__main__':
    def nothing(*arg):
        pass

h1_old = 0
s1_old = 0
v1_old = 0
h2_old = 0
s2_old = 0
v2_old = 0

cv2.namedWindow( "result" ) # создаем главное окно
cv2.namedWindow( "settings" ) # создаем окно настроек

cam = cv2.VideoCapture(0)
# создаем 6 бегунков для настройки начального и конечного цвета фильтра
cv2.createTrackbar('h1', 'settings', 0, 255, nothing)
cv2.createTrackbar('s1', 'settings', 0, 255, nothing)
cv2.createTrackbar('v1', 'settings', 0, 255, nothing)
cv2.createTrackbar('h2', 'settings', 255, 255, nothing)
cv2.createTrackbar('s2', 'settings', 255, 255, nothing)
cv2.createTrackbar('v2', 'settings', 255, 255, nothing)
crange = [0,0,0, 0,0,0]

while True:
```

```
        flag, img = cam.read()
        # Перевод изображения в HSV
        hsv = cv2.cvtColor(img, cv2.COLOR_BGR2HSV)

        # считываем значения бегунков
        h1 = cv2.getTrackbarPos('h1', 'settings')
        s1 = cv2.getTrackbarPos('s1', 'settings')
        v1 = cv2.getTrackbarPos('v1', 'settings')
        h2 = cv2.getTrackbarPos('h2', 'settings')
        s2 = cv2.getTrackbarPos('s2', 'settings')
        v2 = cv2.getTrackbarPos('v2', 'settings')

        # формируем начальный и конечный цвет фильтра
        h_min = np.array((h1, s1, v1), np.uint8)
        h_max = np.array((h2, s2, v2), np.uint8)

        # накладываем фильтр на кадр в модели HSV
        thresh = cv2.inRange(hsv, h_min, h_max)

        cv2.imshow('result', thresh)

        ch = cv2.waitKey(5)

        if h1 != h1_old or s1 != s1_old or v1 != v1_old \
        or h2 != h2_old or s2 != s2_old or v2 != v2_old :
            print ("resault: (%d,%d,%d),(%d,%d,%d)"%(h1,s1,v1,h2,s2,v2))

        h1_old = h1
        s1_old = s1
        v1_old = v1
        h2_old = h2
        s2_old = s2
        v2_old = v2

        if ch == 27:
            break

    cam.release()
    cv2.destroyAllWindows()
```

Интерфейс программы выглядит так:

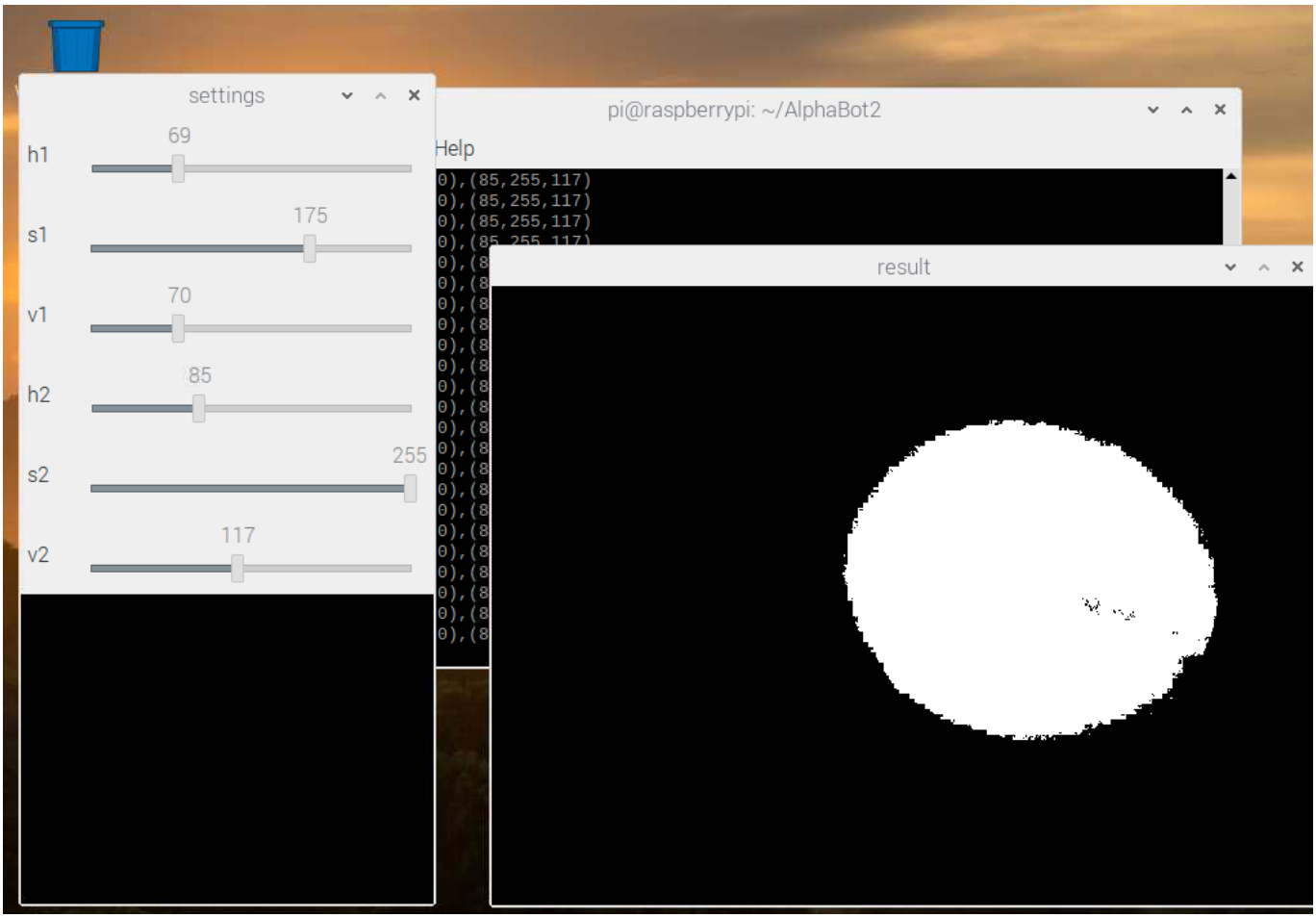


Суть происходящего в том, что для того чтобы определить объект на фото или видео нам нужно выделить его контур, а для этого необходимо знать его цвет. Также для более удобной работы мы переводим цвет в HSV (тон, насыщенность, значение) представление.



Наша задача подобрать минимальное и максимальное значения для трех параметров - H, S, V. Сначала нужно подобрать нижнюю границу параметра тона H (h1). По сути это и есть выбор того цвета, который мы хотим распознать. Для этого нужно двигать первый ползунок до тех пор пока круг не станет видемым. Потом подгоняем максимальное значение H (h2) так, чтобы круг все еще отчетливо был виден, но при этом эти два ползунка были максимально близки. Потом также подгоняем второй параметр S - это насыщенность, чем больше этот параметр, тем «чище» цвет, а чем ближе этот параметр к нулю, тем ближе цвет к нейтральному серому. Третий V - это яркость.

В итоге должно получится что-то похожее на это:



Далее нужно сохранить последнее значение распечатанное в консоль, там будет что-то похожее на это:

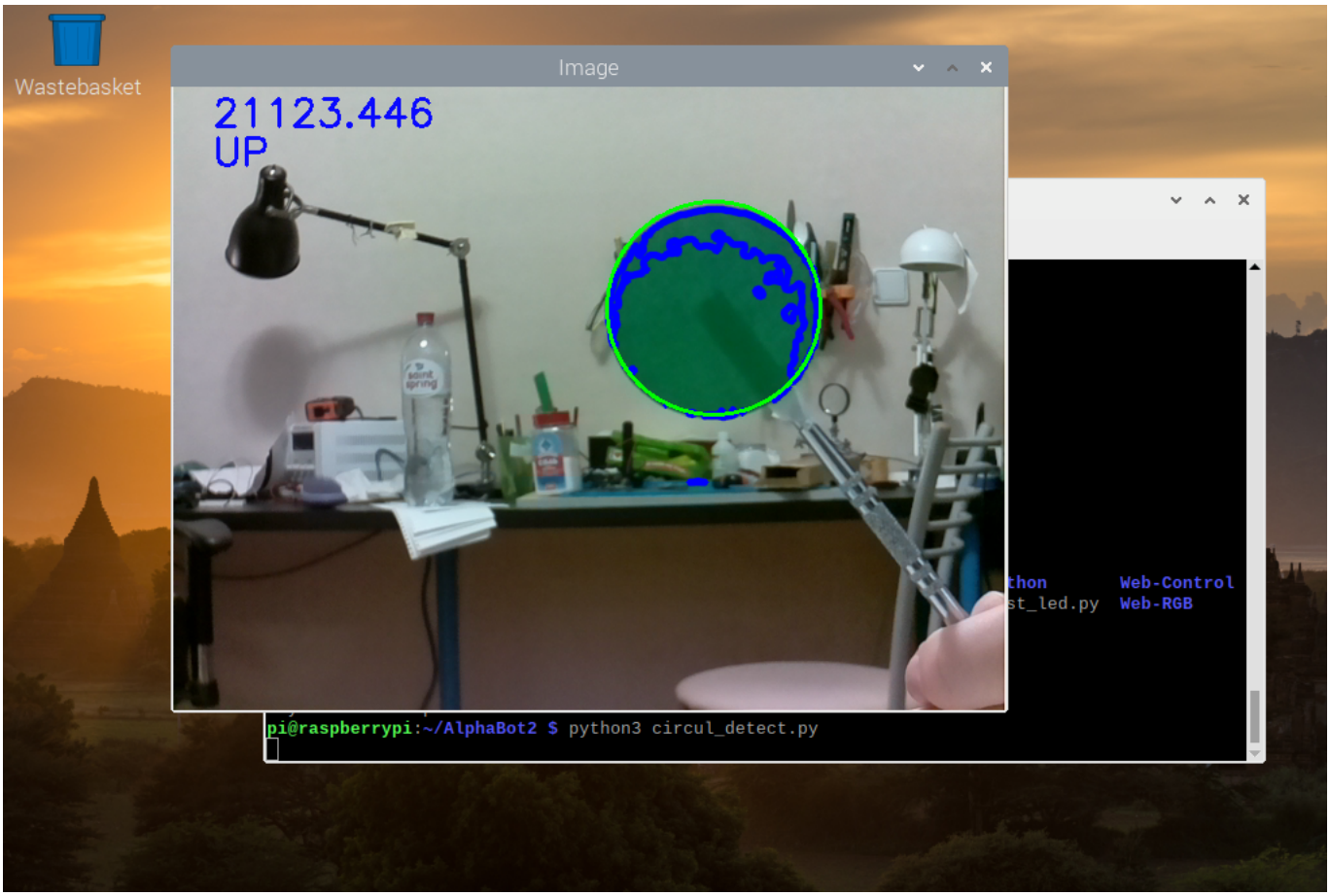
```
(96, 175,70),(85,255,117)
```

Эти числа нам потребуются далее для работы с распознаванием объекта.

Далее рассмотрим пример который распознает круг.

```
import cv2
import numpy as np

# Подключаемся к камере
cam = cv2.VideoCapture(0)

while True:
    _, frame = cam.read()
    # Перевод изображения в HSV
    hsv = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)
    # Немного размываем изображение
    hsv = cv2.blur(hsv,(5,5))

    # Задаем параметр из прошлого примера
    mask = cv2.inRange(hsv, (78,154,93),(86,224,255))
```

```
    # С помощью специального алгоритма ищем контур обьекта
    (contours, hierarchy) = cv2.findContours(mask, cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_SIMPLE)
    # Отрисовывем найденый контур чтобы его было видно на экране
    cv2.drawContours(frame, contours, -1, (255,0,0), 3, cv2.LINE_AA,
hierarchy, 1 )

    max_radius = 0
    center = (0,0)

    # Находим контур с максимальным радиусом
    for contour in contours:
        (x,y),radius = cv2.minEnclosingCircle(contour)
        if max_radius < int(radius):
            max_radius = int(radius)
            center = (int(x),int(y))

    # Описываем максимальный контур кругом
    frame = cv2.circle(frame,center,max_radius,(0,255,0),2)

    S = 3.1415 * max_radius * max_radius
    # Выводим на эран площадь круга
    cv2.putText(frame, str(S), (30, 30),cv2.FONT_HERSHEY_SIMPLEX, 1, (255,
10, 10), 2)

    # Выводим сообщение если круг слишком близко или далеко
    if S > 100:
        if S > 10000:
            cv2.putText(frame, "UP", (30, 60),cv2.FONT_HERSHEY_SIMPLEX, 1,
(255, 10, 10), 2)
        elif S < 5000:
            cv2.putText(frame, "DOWN", (30, 60),cv2.FONT_HERSHEY_SIMPLEX,
1, (255, 10, 10), 2)

    # Промежуточные изображения
    #cv2.imshow("Image with opening", mask)
    #cv2.imshow("Image with closing", hsv)

    # выводим то что получилось вместе с текстом и контуром
    cv2.imshow("Image", frame)

    k = cv2.waitKey(2)
    if k == 27:
        cv2.destroyAllWindows()
        break
```



Изображение, полученное с камеры, проходит несколько этапов преобразования для того, чтобы мы могли распознать на нем объект. Сначала мы преобразуем изображение в HSV формат, который мы использовали в предыдущим примере. Потом с помощью алгоритма, определяющего контур, ищем объект нужного цвета. Вот тут и нужны параметры полученные в прошлом примере, из-за того что у вас может отличатся освещение или камера то эти параметры всегда нужно подбирать экспериментально. Этот контур можно увидеть на изображении выше, он состоит из синих точек. Далее мы ищем две самые удаленные друг от друга точки и строим вокруг них окружность, определяем площадь этой окружности, выводя сообщение о том что она слишком близко или далеко от камеры. Здесь можно добавить, например, движение робота за объектом если он далеко и от него если он близко, также можно определить положение объекта на экране, таким образом заставив робота следовать за объектом.

Машинное обучение

Тут в качестве примера я предложу вам обучить свою нейронную серь на открытом датасете который содержит тысячи изображений цифр - MNIST. Этот датасет уже встроен в библиотеку keras.

Для того чтобы использовать нейронную сеть ее нужно сначала обучить, для этого я предлагаю использовать сервис google colaboratory - https://colab.research.google.com/notebooks/intro.ipynb, потому что обучение сетей требует значительных вычислительных мощностей, а компания Google готова нам предоставить их бесплатно в образовательных целях. Данный сервис представляет из себя страницу jupiter notebook, код в которой исполняется поблочно. Давайте рассмотрим блоки, которые нам необходимы для получения обученной сети:

1. Импорт библиотек

```
import numpy as np
from tensorflow import keras
from tensorflow.keras import layers
```

2. Скачивание датасета и преобразование его в необходимый формат:

```
# Параметры модели и данных
num_classes = 10
input_shape = (28, 28, 1)

# Разделяем данные на тестовые и обучающие
(x_train, y_train), (x_test, y_test) = keras.datasets.mnist.load_data()

# Масштабируем изображения до диапазона цветов 0..1
x_train = x_train.astype("float32") / 255
x_test = x_test.astype("float32") / 255
# Меняем размер изображения (28, 28, 1)
x_train = np.expand_dims(x_train, -1)
x_test = np.expand_dims(x_test, -1)
print("x_train shape:", x_train.shape)
print(x_train.shape[0], "train samples")
print(x_test.shape[0], "test samples")

y_train = keras.utils.to_categorical(y_train, num_classes)
y_test = keras.utils.to_categorical(y_test, num_classes)
```

3. Создание нейронной сети, прописываем слои из которых мы хотим чтобы состояла наша нейросеть:

```
model = keras.Sequential(
    [
        keras.Input(shape=input_shape),
        layers.Conv2D(32, kernel_size=(3, 3), activation="relu"),
        layers.MaxPooling2D(pool_size=(2, 2)),
        layers.Conv2D(64, kernel_size=(3, 3), activation="relu"),
        layers.MaxPooling2D(pool_size=(2, 2)),
        layers.Flatten(),
        layers.Dropout(0.5),
        layers.Dense(num_classes, activation="softmax"),
    ]
)

model.summary()
```

В результате можно увидить структуру созданой сети:

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 26, 26, 32)        320
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 13, 13, 32)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 11, 11, 64)        18496
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 5, 5, 64)          0
_________________________________________________________________
flatten (Flatten)            (None, 1600)              0
_________________________________________________________________
dropout (Dropout)            (None, 1600)              0
_________________________________________________________________
dense (Dense)                (None, 10)                16010
=================================================================
Total params: 34,826
Trainable params: 34,826
Non-trainable params: 0
_________________________________________________________________
```

4. Обучение нейросети. Это долгий процесс, у меня заняло около 30 мин:

```
batch_size = 128
epochs = 15

model.compile(loss="categorical_crossentropy", optimizer="adam", metrics=
["accuracy"])

model.fit(x_train, y_train, batch_size=batch_size, epochs=epochs,
validation_split=0.1)
```

5. Проверка точности и сохранение обученой сети:

```
score = model.evaluate(x_test, y_test, verbose=0)
model.save('mnist_trained_model.h5')
print("Test loss:", score[0])
print("Test accuracy:", score[1])
```

После этого нужно будет перести файл `mnist_trained_model.h5` на вашу Raspberry Pi.

Подготовим Raspberry Pi к запуску нейросети. В первую очередь нужно установить еще некоторые пакеты:

```
sudo apt install python3-h5py
pip3 install keras
wget https://github.com/lhelontra/tensorflow-on-
arm/releases/download/v2.4.0/tensorflow-2.4.0-cp37-none-linux_armv7l.whl
pip3 install tensorflow-2.4.0-cp37-none-linux_armv7l.whl
```

Это сработает только для Python 3.7. В случае установленной более новой версии найдите подходящую версию tensorflow здесь (https://github.com/lhelontra/tensorflow-on-arm/releases) или используйте `pyenv`, также версия tensorflow должна совпадать с версией на google colaboratory, иначе ничего не заработает.

Теперь запустим следующий скрипт:

```
import cv2
import numpy as np
import keras

def find_number(frame):
    # функция выполняет предобработку изображения
    # находит область с цифрой по контурам
    # далее в цикле перебераем найденные контуры применяя фильтры
    # если контур найден, то меняем разрешение на 28*28
    # выполняем бинаризацию, делим на 255 и инвертируем вычитая 1
    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
    blur = cv2.GaussianBlur(gray,(3,3), 0)
    edges = cv2.Canny(blur, 50, 100)
    contours, _ = cv2.findContours(edges, cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_SIMPLE)
    img2 = np.zeros((28, 28), np.uint8)
    img_rec = np.zeros((1, 28, 28, 1), np.uint8)
    fail = True

    for contr in contours:

        if cv2.contourArea(contr) < 500:
            continue
        x,y,w,h = cv2.boundingRect(contr)
        img2 = frame[y-5:y+h+5, x-5:x+w+5]

        # Проверка на некорректное изображение
        if img2.shape[0] <= 0 or img2.shape[1]<= 0:
            continue

        #Фильтер по вертикальности рамки
        if img2.shape[0] < img2.shape[1]:
            continue

        img2 = cv2.resize(img2, (28, 28))
        img2 = cv2.cvtColor(img2, cv2.COLOR_BGR2GRAY)
```

```
        # th3 =
cv2.adaptiveThreshold(img2,255,cv2.ADAPTIVE_THRESH_GAUSSIAN_C,\
        #     cv2.THRESH_BINARY,11,2)

        _, th3 = cv2.threshold(img2,127,255,cv2.THRESH_BINARY)

        # Фильтруем по следнему цвету, должно быть много белого
        avg_color_per_row = np.average(th3, axis=0)
        avg_color = np.average(avg_color_per_row, axis=0)
        if avg_color < 180:
            continue

        # Если все норм то рисуем рамки на исходном изображении.
        img = cv2.rectangle(frame,(x,y),(x+w,y+h),(0,255,0),2)

        cv2.imshow("img2", img2)
        cv2.imshow("th3", th3)

        img_rec = th3/255.0
        img_rec = 1 - img_rec
        img_rec = img_rec.reshape((1,28,28,1))

        fail = False
    return img2, img_rec, fail



if __name__ == '__main__':
    cam = cv2.VideoCapture(0)
    model = keras.models.load_model("mnist_trained_model.h5")
    try:
        while True:
            _, frame = cam.read()
            img_show, img_rec, fail = find_number(frame)
            if fail == False:
                result = model.predict_classes([img_rec])
                cv2.putText(frame,str(result[0]),(10,460),
cv2.FONT_HERSHEY_SIMPLEX, 1,(0,0,255),2,cv2.LINE_AA)


            cv2.imshow("Image", frame)

            k = cv2.waitKey(2)
            if k == 27:
                #cv2.imwrite('diff_robot_nn/nine.jpg', img_show)
                cv2.destroyAllWindows()
                break
    except Exception as e:
        cam.release()
        cv2.destroyAllWindows()
        print(e)
```

Распознование можно разделить на два этапа, первый - это предобработка изображения, а второй - это само распознавание.

На первом этапе (который выделен в отдельную функцию *find_number(frame)*) мы ищем все контура изображений (тоже самое, что мы делали в предыдущем примере для распознавания цветного круга), потом перебераем контуры, отсеивая неподходящие. Первым делом мы отбрасываем контуры маленькой площади, затем выбрасываем контуры, вытянутые по горизонтали (так как все цифры вытянуты по вертикали), потом переводим изображение в черно-белый формат, сжимаем до размера 28x28 и делаем его монохромным. Это формат необходимый для обученной нами ранее сетки. Далее мы применяем еще один фильтр, суть которого заключается в том, что мы отбрасываем изображения, в которых черного цвета больше чем белого. И после этого отправляем подготовленное изображение в нейронную сеть, а она выдает предполагаемое число. Тут необходимо помнить, что данный датасет собран на числах написанных ручкой в маленькой тетрадной клетке, из-за чего в реальность точность будет около 75%, несмотря на то что на тестовых данных было 99%. Также озображение должно быть хорошо освещено чтобы корректно распознатся.