

1. Введение в основной микроконтроллер

1.1 Что такое микросхема K210?

Микросхема K210 представляет собой микроконтроллер, построенный на архитектуре RISC-V, и обладает рядом отличительных особенностей. Ключевой характеристикой данной архитектуры является наличие аппаратного ускорителя нейронных сетей KPU (Kendryte Processing Unit), предназначенного для высокопроизводительных вычислений свёрточных нейронных сетей.

В контексте вычислений искусственного интеллекта на уровне MCU производительность K210 является весьма высокой. Согласно официальным данным производителя (Canaan), производительность KPU достигает 0,8 TFLOPS. Для сравнения:

Raspberry Pi 4B обладает вычислительной мощностью менее 0,1 TFLOPS;

Jetson Nano, ориентированный на нейросетевые вычисления и оснащённый 128 CUDA-ядрами, обеспечивает около 0,47 TFLOPS K210 Module_1.

Помимо высокой производительности KPU, микросхема K210 поддерживает технологию FPIOA (Field Programmable IO Array) — программируемую матрицу ввода-вывода, позволяющую свободно сопоставлять периферийные функции с любыми физическими выводами. Это существенно упрощает разводку GPIO и назначение выводов при разработке.

K210 оснащён двухъядерным 64-битным процессором RISC-V, при этом каждое ядро имеет встроенный блок вычислений с плавающей запятой (FPU), что позволяет выполнять независимые операции с вещественными числами.

Для задач машинного зрения и аудиообработки микросхема дополнительно содержит:

KPU — аппаратный ускоритель свёрточных нейронных сетей;

APU (Audio Processing Unit) — ускоритель обработки сигналов микрофонного массива, обеспечивающий высокую производительность аудиоанализа.

Кроме того, K210 включает аппаратный ускоритель быстрого преобразования Фурье (FFT), что позволяет эффективно выполнять сложные спектральные вычисления.

С точки зрения безопасности микросхема оснащена аппаратными ускорителями криптографических алгоритмов AES и SHA-256, обеспечивающими защиту пользовательских данных.

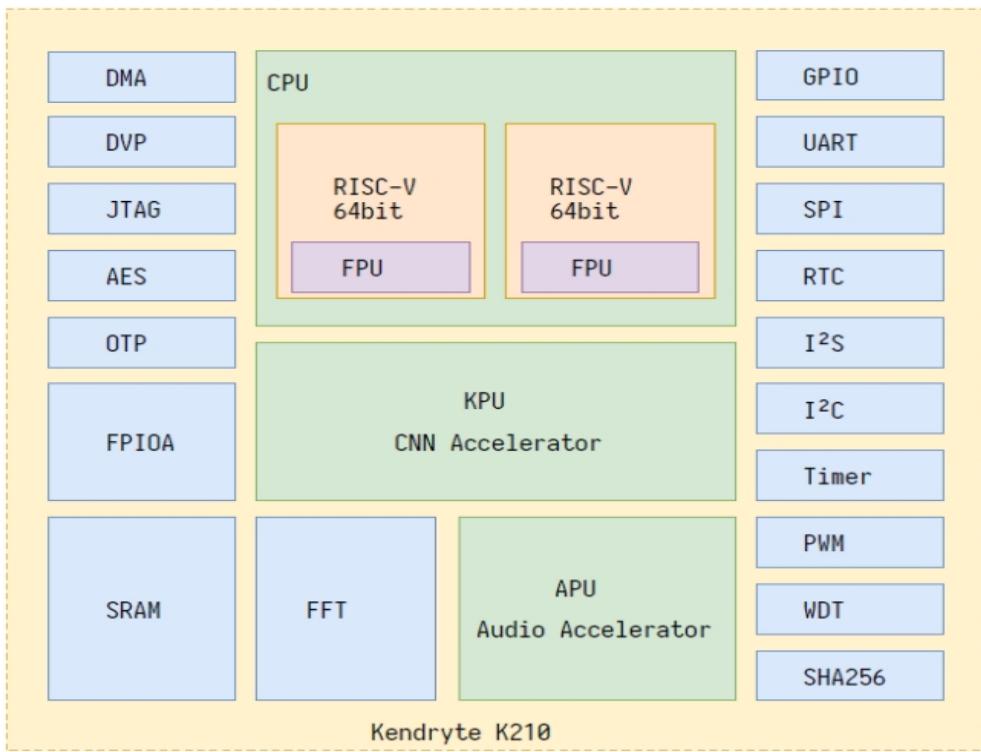
Как микроконтроллер, K210 обладает богатым набором периферийных модулей, включая: DVP, JTAG, OTP, FPIOA, GPIO, UART, SPI, RTC, I2S, I2C, WDT, TIMER, PWM, что покрывает большинство типовых потребностей MCU-приложений.

Микросхема содержит 8 МБ высокопроизводительной SRAM, из которых:

2 МБ выделены для задач искусственного интеллекта;

6 МБ используются для программного кода.

Также предусмотрен внешний интерфейс для FLASH-памяти и высокоэффективный механизм передачи данных DMA, обеспечивающий высокую пропускную способность. Структурная схема архитектуры K210 приведена на рисунке в документации и предназначена для справочного использования



1.2 Что такое система команд RISC-V?

RISC-V — это открытая архитектура набора команд (ISA), основанная на принципах RISC (Reduced Instruction Set Computer). Буква «V» обозначает пятую версию архитектуры, разработанную в Калифорнийском университете в Беркли на основе четырёх предыдущих поколений.

Проект RISC-V был инициирован в 2010 году, и за более чем десятилетие активного развития при участии мирового сообщества архитектура получила широкое распространение. В настоящее время RISC-V активно используется, и ожидается дальнейший рост числа микросхем на её основе K210 Module_1.

Основные характеристики RISC-V:

Полная открытость

Любая компания может бесплатно использовать архитектуру RISC-V для разработки и производства собственных микросхем без лицензионных отчислений.

Допускается расширение набора команд под собственные задачи без обязательного раскрытия таких расширений.

Простота архитектуры

В отличие от x86 и ARM, RISC-V не требует обратной совместимости, что делает базовый набор

Удобство портирования операционных систем

Архитектура поддерживает разделение привилегированных и пользовательских режимов выполнения, что обеспечивает стабильность и безопасность ОС. Наличие спецификаций пользовательских и привилегированных инструкций упрощает портирование Linux и Unix-подобных систем.

Модульность

RISC-V позволяет формировать конфигурации под конкретные задачи.

Например:

RV32IC — для маломощных встраиваемых систем;

RV32IMFDC — для высокопроизводительных систем с ОС и многозадачностью.

Полноценная цепочка инструментов

Благодаря многолетней поддержке сообщества и фонда RISC-V доступен полный набор средств разработки (toolchain), необходимый для компиляции, отладки и взаимодействия программного обеспечения с процессором.

1.2. Введение в модуль визуального распознавания K210

Общее описание модуля

В данном разделе приводится общее описание модуля, его конструктивных элементов и внешнего вида.

1.2.1 Лицевая сторона модуля

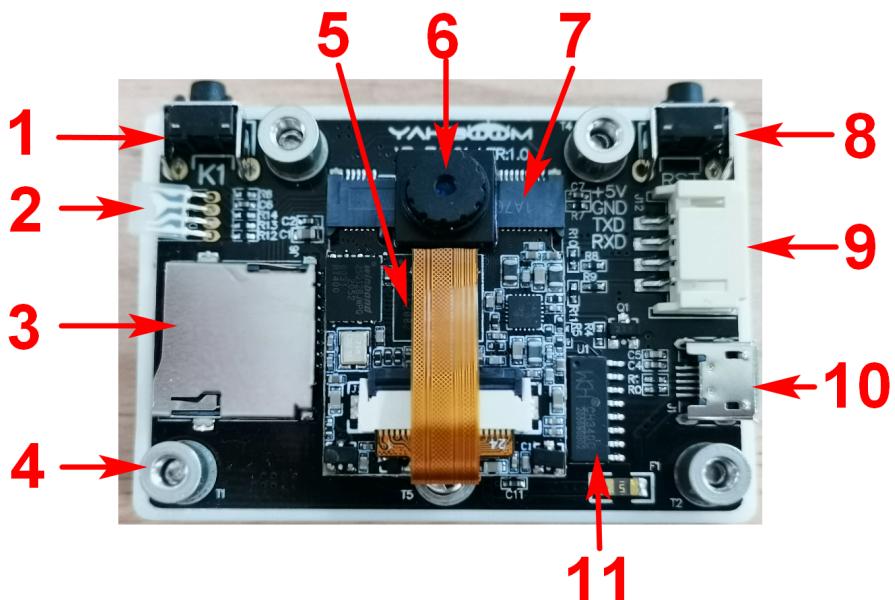


Лицевая сторона модуля

Лицевая сторона модуля представляет собой 2,0-дюймовый ёмкостный сенсорный экран с разрешением 320×240 пикселей. Экран используется для отображения результатов работы алгоритмов, интерфейса пользователя и отладочной информации

2.2 Обратная сторона модуля

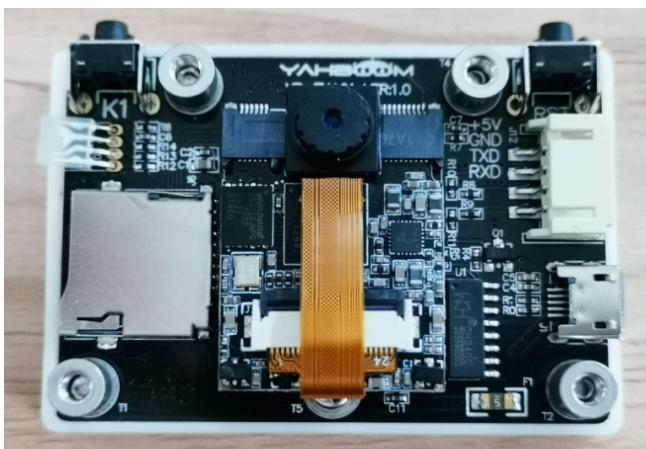
На обратной стороне модуля расположены основные аппаратные компоненты:



1. Кнопка K1 — аппаратно подключена к выводу IO16, функция кнопки может быть задана пользователем.
2. RGB-светодиоды — программируемая индикация красного, зелёного, синего и белого цветов.
3. Слот TF-карты — используется для установки карты памяти TF (контакты карты направлены к плате); предназначен для хранения программ и файлов моделей.
4. Фиксирующие медные стойки — четыре стойки стандарта M3 для механического крепления.
5. Ядро модуля K210 — содержит микросхему K210 и минимальную систему её функционирования.
6. Камера — предназначена для захвата изображений.
7. Интерфейс модуля — соединяет основной модуль K210 с нижней платой.
8. Кнопка RST (Reset) — используется для перезапуска микросхемы K210.
9. Внешний последовательный порт — предназначен для подключения внешних UART-устройств и вывода пользовательских данных.
10. Интерфейс microUSB — используется для загрузки прошивки, отладки через последовательный порт и подключения к среде разработки (IDE).
11. Микросхема CH340 — преобразует USB-сигнал в TTL-уровни при подключении через microUSB .

2.3 Физический вид изделия

В подразделе представлены фотографии модуля с различных ракурсов, демонстрирующие внешний вид лицевой и обратной сторон, расположение камеры, разъёмов и элементов крепления



3.1 Hardware pin assignment

Hardware IO port	function
IO_RST	RESET K210
IO_1	Unused
IO_2	Unused
IO_3	Unused
IO_4	ISP_TX, connect microUSB TX
IO_5	ISP_RX, connect microUSB RX
IO_6	RXD, The receiving pin(RX) of the expansion interface
IO_7	Unused
IO_8	TXD, The send pin(TX) of the expansion interface
IO_9	Unused
IO_10	Unused
IO_11	Unused
IO_12	Unused
IO_13	Unused
IO_14	Unused
IO_15	Unused
IO_16	BOOT
IO_17	connect key K1
IO_18	Unused
IO_19	Unused
IO_20	Unused
IO_21	Unused
IO_22	FT_INT, Connect the interrupt pin to the touch screen
IO_23	LCD-RST, Connect to the reset pin of the touch screen
IO_24	I2C-SCL, Connect the I2C-SCL pin of the touchpad
IO_25	I2C-SDA, Connect the I2C-SDA pin of the touchpad
IO_26	RGB-G, Connect to the G pin of the RGB lamp
IO_27	RGB-R, The R pin is connected to the RGB lamp
IO_28	LCD-WR, Connect to the WR(CLK) pin of the LCD
IO_29	RGB-B, The B pin is connected to the RGB lamp
IO_30	LCD-CS, The CS pin is connected to the LCD
IO_31	LCD-RS, The RS(DC) pin connected to the LCD
IO_32	SPI-CLK, The CLK pin connected to the TF card slot
IO_33	SPI-MISO, The MISO pin connected to the TF card slot
IO_34	SPI-CS, The CS pin connected to the TF card slot
IO_35	SPI-MOSI, The MOSI pin connected to the TF card slot
IO_36	Unused
IO_37	Unused
IO_38	Unused
IO_39	Unused
IO_40	DVP-SDA, Connect to the camera DVP-SDA pin
IO_41	DVP-SCL, Connect to the camera DVP-SCL pin
IO_42	DVP-RST, Connect to the camera DVP-RST pin
IO_43	DVP-VSYNC, Connect to the camera DVP-VSYNC pin
IO_44	DVP-PWDN, Connect to the camera DVP-PWDN pin
IO_45	DVP-HSYNC, Connect to the camera DVP-HSYNC pin
IO_46	DVP-XCLK, Connect to the camera DVP-XCLK pin
IO_47	DVP-PCLK, Connect to the camera DVP-PCLK pin

Fixed IO port	function
SPI0_D7	LCD-D7, Connect to the LCD-D7 pin
SPI0_D6	LCD-D6, Connect to the LCD-D6 pin
SPI0_D5	LCD-D5, Connect to the LCD-D5 pin
SPI0_D4	LCD-D4, Connect to the LCD-D4 pin
SPI0_D3	LCD-D3, Connect to the LCD-D3 pin
SPI0_D2	LCD-D2, Connect to the LCD-D2 pin
SPI0_D1	LCD-D1, Connect to the LCD-D1 pin
SPI0_D0	LCD-D0, Connect to the LCD-D0 pin
DVP_D7	DVP-D7, Connect to the DVP-D7 pin
DVP_D6	DVP-D6, Connect to the DVP-D6 pin
DVP_D5	DVP-D5, Connect to the DVP-D5 pin
DVP_D4	DVP-D4, Connect to the DVP-D4 pin
DVP_D3	DVP-D3, Connect to the DVP-D3 pin
DVP_D2	DVP-D2, Connect to the DVP-D2 pin
DVP_D1	DVP-D1, Connect to the DVP-D1 pin
DVP_D0	DVP-D0, Connect to the DVP-D0 pin

3.2 Software GPIO allocation

GPIOHS Default	function	Binding hardware IO
GPIOHS31	LCD-RS, LCD read and write signal pin	IO_31
GPIOHS30	LCD_RST, LCD reset pin	IO_23
GPIOHS29	SPI-CS, TF card SPI slice selection pin	IO_34
GPIOHS25	RGB-B, RGB light blue pin	IO_29
GPIOHS24	RGB-G, RGB light green pin	IO_26
GPIOHS23	RGB-R, RGB lamp red pin	IO_27

4. Введение в MicroPython

MicroPython — что это такое?

MicroPython — это компактная и эффективная реализация языка программирования Python 3, включающая подмножество стандартной библиотеки Python, оптимизированная для работы на микроконтроллерах и в средах с ограниченными ресурсами.

MicroPython содержит множество продвинутых возможностей, таких как:

интерактивная командная строка (REPL),
целые числа произвольной точности,
замыкания (closures),
генераторы и генераторные выражения,
обработка исключений и другие современные возможности языка Python.

При этом MicroPython остаётся чрезвычайно компактным:
для работы требуется всего около 256 КБ памяти под код и 16 КБ оперативной памяти.

Основная цель MicroPython — обеспечить максимальную совместимость с обычным Python, чтобы разработчик мог легко переносить код с настольного компьютера на микроконтроллер или встроенную систему с минимальными изменениями.

FAQ — Часто задаваемые вопросы по модулю K210

1. Как подаётся питание на модуль K210?

Ответ:

Рабочее напряжение питания модуля K210 составляет 5 В. Питание может подаваться:
через разъём microUSB;
через внешний последовательный порт.

2. Может ли модуль K210 работать без TF-карты? Есть ли ограничения на TF-карту?

Ответ:

Да, программа может работать и без TF-карты.

Из-за ограниченного объёма встроенной памяти K210, TF-карта в основном используется для хранения файлов моделей нейронных сетей.

Если задачи не связаны с AI-распознаванием, использование TF-карты необязательно. В этом случае программный код сохраняется во встроенной Flash-памяти микросхемы K210.

Требования к TF-карте:

файловая система FAT32;

не все TF-карты совместимы — рекомендуется ориентироваться на практическое тестирование.

3. При подключении к CanMV IDE программа часто обрывается — в чём причина?

Ответ:

Модуль K210 не поддерживает нативное USB-соединение и использует USB–UART преобразователь.

При передаче видеопотока для предпросмотра возникает большой объём данных, что может приводить к нестабильному соединению и разрывам.

Решение:

В CanMV IDE нажмите кнопку Disable Camera Preview (в правом верхнем углу), чтобы отключить предпросмотр камеры.

4. K210 подключен по USB, но CanMV IDE не видит устройство. Что делать?

Ответ:

Проверьте:

выбран ли правильный COM-порт, соответствующий модулю K210;
в настройках подключения выберите режим Mode-2.

Если ранее на модуль была прошита сторонняя прошивка, необходимо:
 заново прошить заводскую (factory) прошивку,
 затем повторить подключение.

5. После подключения к CanMV IDE кнопка K1 не реагирует. Это неисправность?

Ответ:

Нет.

Кнопка K1 не работает в онлайн-режиме при подключении к CanMV IDE.

Для её использования необходимо:

загрузить программу в файл main.py,

запустить её автономно на модуле.

6. Какой экран установлен на модуле K210?

Ответ:

Модуль K210 оснащён:

ёмкостным сенсорным экраном,

диагональ 2,0 дюйма,

разрешение 320 × 240 пикселей.

7. Почему при подключении к Serial Port Assistant программа не запускается?

Ответ:

При работе с последовательным портом необходимо правильно установить сигналы RTS и DTR.

Если вы используете, например, UartAssist:

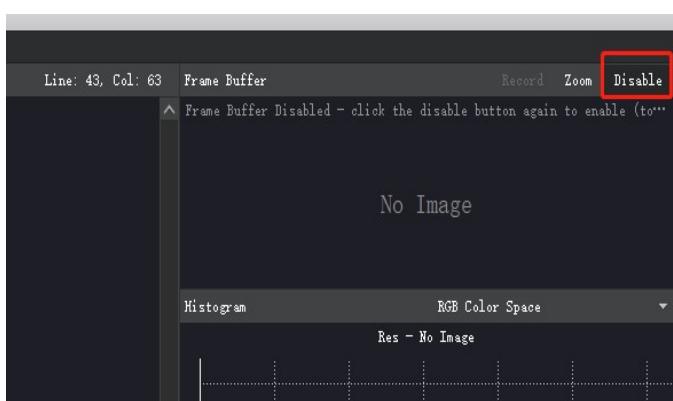
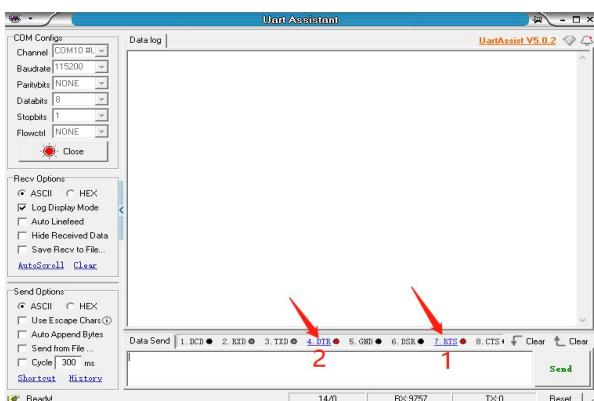
откройте последовательное соединение;

нажмите RTS (пункт 7);

затем нажмите DTR (пункт 4).

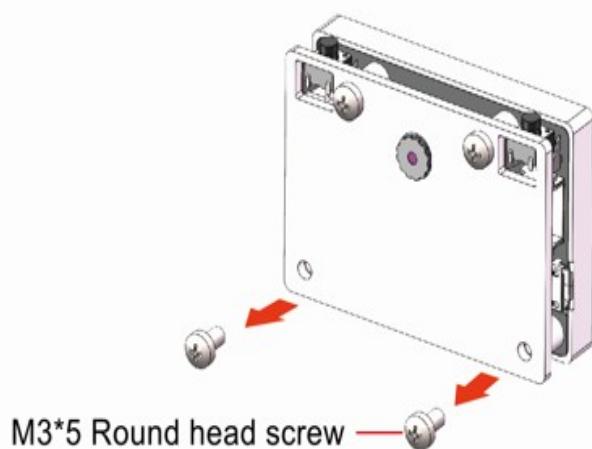
Только после этого программа начнёт выполняться корректно.

Изменение индикатора состояния с красного на чёрный означает нормальный режим работы.



K210 vision recognition module installation bracket steps

1. Unscrew the M3*5mm round head screws under the K210 vision recognition module.



2. Install bracket



2. Настройка среды разработки

1. Установка драйвера CH340 (Windows 10 x64)

Этот раздел показывает установку драйвера CH340 на примере Windows 10 x64. Если драйвер CH340 у вас уже установлен, эти шаги можно пропустить.

1.1. Установка драйвера последовательного порта CH340

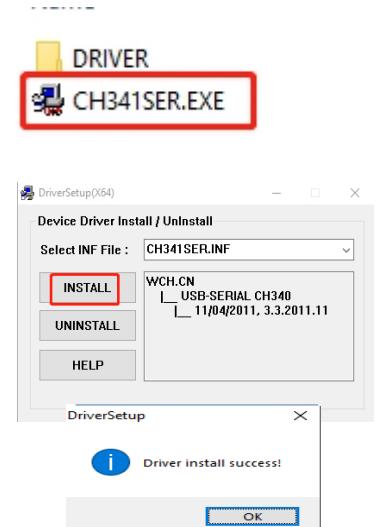
1. Распакуйте архив с драйвером CH340.

Обычно он называется примерно так: Uart drive (CH340).zip (название может немного отличаться).

2. Откройте распакованную папку и дважды щёлкните по файлу: CH341SER.EXE

3. В открывшемся окне нажмите кнопку Install (Установить).

4. После завершения появится сообщение о том, что установка прошла успешно.



Проверка, что драйвер установлен и порт появился

1. Подключите плату K210 Development Board к компьютеру с помощью кабеля Type-C.

2. Откройте Диспетчер устройств:

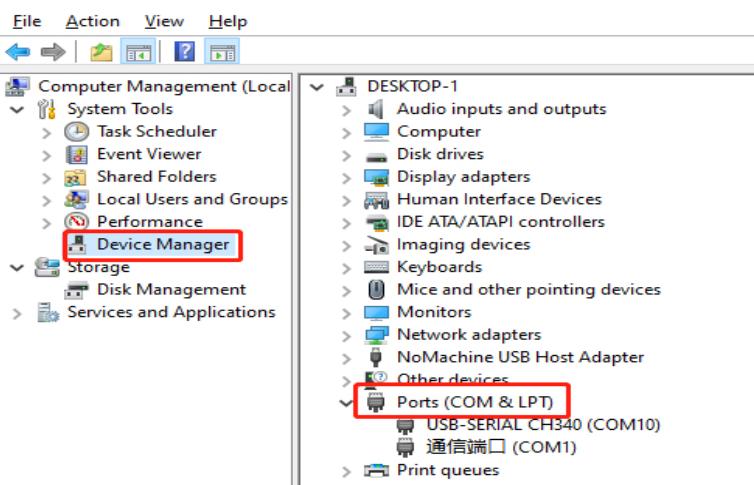
ПКМ по рабочему столу → Этот компьютер → Свойства

→ Диспетчер устройств (или Win + X → Диспетчер устройств)

3. Найдите раздел: Ports (COM & LPT) / Порты (COM и LPT)

Там должен появиться порт вида CH340 / USB-SERIAL CH340, например:

USB-SERIAL CH340 (COM3)



Если он появился — драйвер установлен правильно, и плата корректно определяется системой.

Примечание

Номер COM-порта (COM3, COM4, COM7 и т.д.) на разных компьютерах может отличаться — это нормально. Главное, чтобы система распознала устройство как CH340.

1.2. Открытие Serial Port Assistant (UartAssist)

1. Запустите программу UartAssist (Serial Port Assistant).

2. В настройках порта установите параметры:

Serial Port (COM): выберите COM-порт платы K210 Development Board (CH340)

Baud rate (скорость): 115200

Parity (чётность): NONE

Data bits (биты данных): 8

Stop bits (стоп-биты): 1

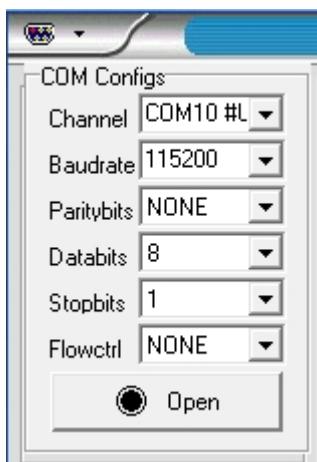
Flow Control (управление потоком): NONE

3. Нажмите Open.

После открытия:

кнопка Open обычно меняется на Off/Close (или аналогичную),

индикатор (иконка) меняет цвет с чёрного на красный, что означает: порт открыт.



Включение DTR и RTS и перезапуск платы

1. В правом нижнем углу интерфейса UartAssist найдите переключатели:

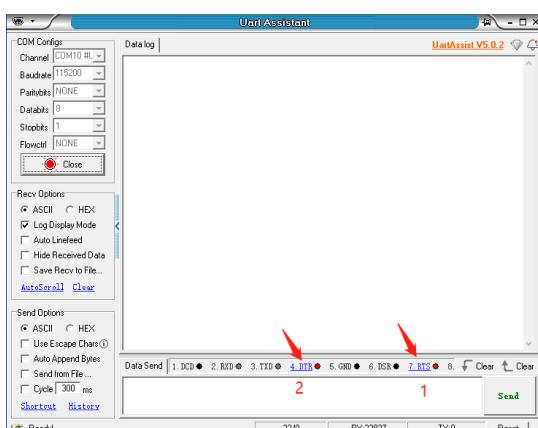
4. DTR

7. RTS

2. По умолчанию они красные. Нажмите на 4. DTR и 7. RTS, чтобы они стали зелёными.

3. Нажмите кнопку Reset на плате K210 Development Board.

Важно: если прошивка внутри платы не выводит сообщения в UART, то в окне UartAssist ничего не появится — это нормально.



2. Установка среды разработки CanMV IDE

2.1 Установка CanMV IDE

Среда разработки CanMV IDE распространяется через официальный репозиторий GitHub. В зависимости от используемой операционной системы необходимо скачать соответствующий установочный пакет.

Официальная страница загрузки CanMV IDE:

https://github.com/kendryte/canmv_ide/releases

Nov 21, 2022
kendryte747
v2.9.2-1 [Latest]
Add update examples from github/gitee, user should install git to system PATH.
sha2
737ef5835faeed9be89db1a2e6d9557640717f6563931a21f0d4962292e7d97d canmv-ide-linux-x86_64-v2.9.2-1-g626670d.run
2d668984080c7a03fd3793cabdf3d96c8745ee539cbf9021e9e43a219c51f2b2 canmv-ide-mac-2.9.2-1-g626670d.dmg
5a111b83f2fc3802b29e2bcd802cc0809a70e5d7e79ee65362c3a641916e36cc canmv-ide-windows-v2.9.2-1-g626670d.exe

▼ Assets 5

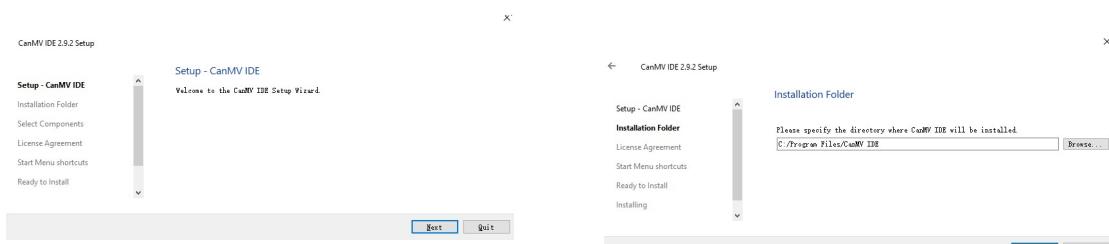
canmv-ide-linux-x86_64-v2.9.2-1-g626670d.run	136 MB	Nov 21, 2022
canmv-ide-mac-2.9.2-1-g626670d.dmg	137 MB	Nov 21, 2022
canmv-ide-windows-v2.9.2-1-g626670d.exe	104 MB	Nov 21, 2022
Source code (zip)		Nov 21, 2022
Source code (tar.gz)		Nov 21, 2022

В качестве примера ниже рассматривается установка для Windows 10.

После загрузки файла вида

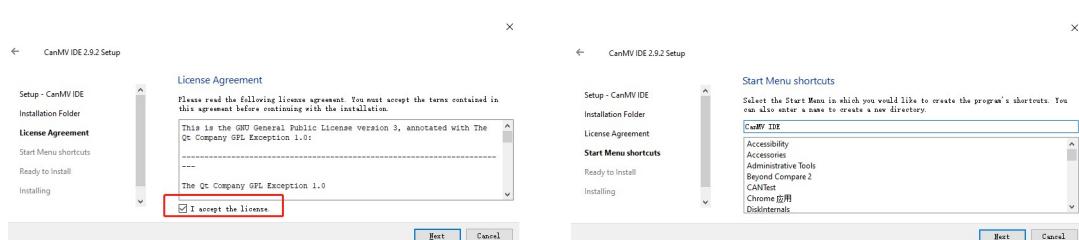
canmv-ide-windows-vx.x.x-gxxxxxx.exe

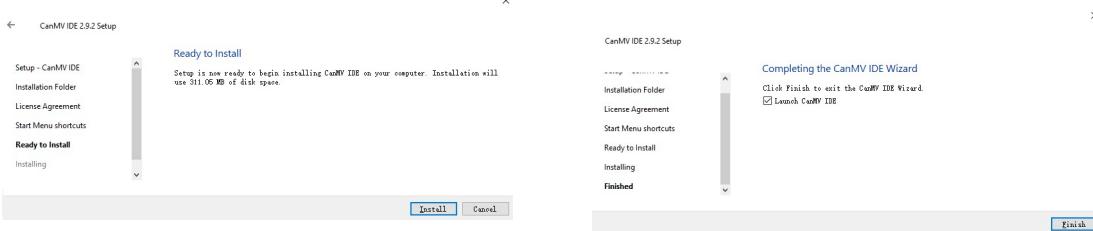
необходимо дважды щёлкнуть по нему для запуска установщика.



В процессе установки:

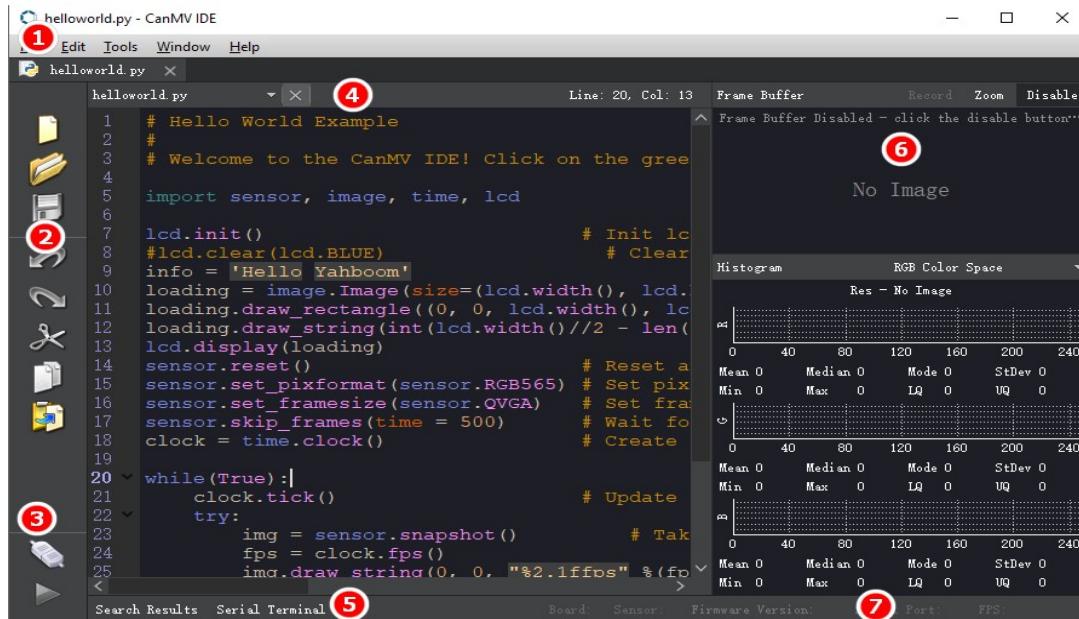
при необходимости можно изменить путь установки;





последовательно подтвердить шаги мастера установки;
дождаться завершения установки программных компонентов.
После окончания установки CanMV IDE будет готова к работе.

2.2 Описание интерфейса CanMV IDE



Интерфейс CanMV IDE состоит из следующих функциональных областей:

Область 1 — строка меню

Содержит основные команды: работа с файлами, инструменты, настройки и справку.

Область 2 — панель быстрых кнопок

Предоставляет быстрый доступ к часто используемым функциям меню (открытие файлов, редактирование и т.п.).

Область 3 — управление подключением устройства

Кнопки подключения платы, запуска и остановки программ.

Область 4 — область редактирования кода

Используется для написания и редактирования программ на MicroPython.

Область 5 — последовательный терминал

Отображает отладочную информацию, выводимую через последовательный порт.

Область 6 — предварительный просмотр изображения

Показывает изображение, получаемое с камеры в реальном времени.

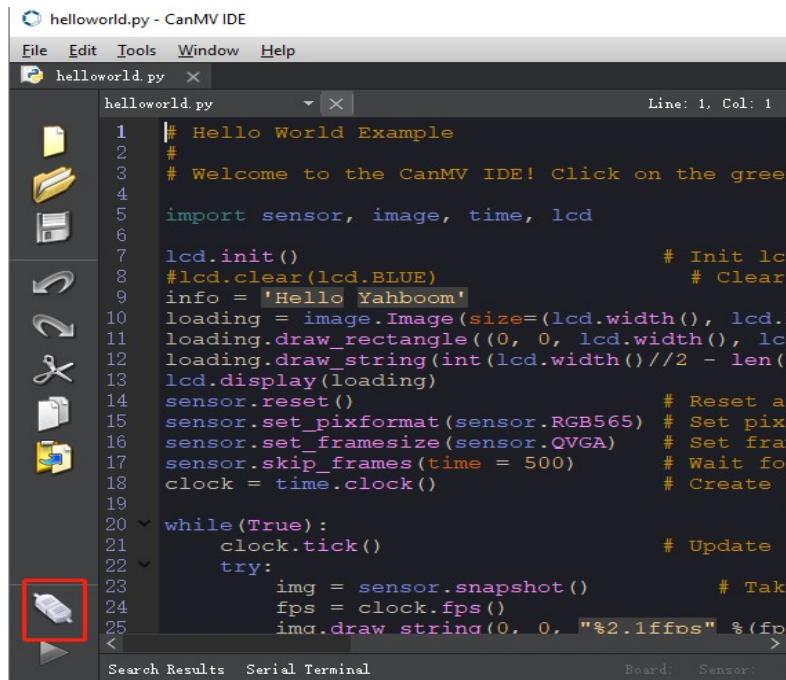
Область 7 — информация о прошивке

Отображает номер последовательного порта и сведения о версии прошивки устройства.

2.3 Подключение устройства

Подключите модуль K210 к USB-порту компьютера с помощью кабеля microUSB.

После этого нажмите кнопку Connect в левом нижнем углу CanMV IDE.



```
# Hello World Example
#
# Welcome to the CanMV IDE! Click on the green play button to run your program.
#
# Import the required modules
import sensor, image, time, lcd

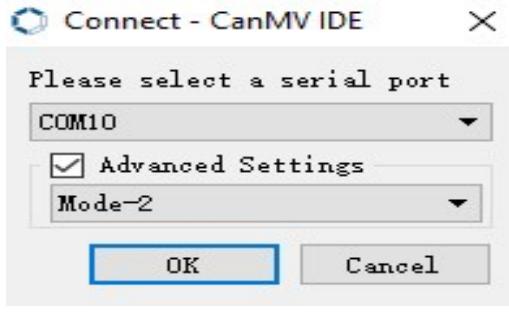
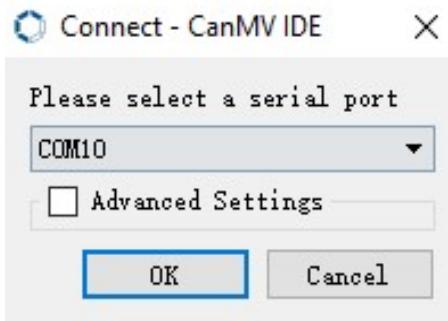
# Initialize the LCD
lcd.init()
# Clear the LCD
info = 'Hello Yahboom'
loading = image.Image(size=(lcd.width(), lcd.height()))
loading.draw_rectangle((0, 0, lcd.width(), lcd.height()), lc
loading.draw_string(int(lcd.width())//2 - len(info)//2, int(lcd.height()//2), info)
lcd.display(loading)

# Reset the sensor
sensor.reset()
# Set pixel format
sensor.set_pixformat(sensor.RGB565)
# Set frame size
sensor.set_framesize(sensor.QVGA)
# Set frame rate
sensor.skip_frames(time = 500)
# Wait for sensor reset
clock = time.clock()

while(True):
    clock.tick()
    try:
        img = sensor.snapshot()
        fps = clock.fps()
        img.draw_string(0, 0, "%2.1ffps" % (fps))
    except:
        pass
```

В появившемся окне выберите последовательный порт, соответствующий модулю K210.

Если доступно несколько портов, необходимо заранее определить нужный номер и затем нажать OK.



Если подключение не устанавливается:

нажмите Advanced settings;

установите параметр Mode в значение 2.

При успешном подключении:

иконка подключения изменит состояние на «подключено»;

кнопка запуска программы станет зелёной.



2.4 Временный запуск программы (Temporary Run)

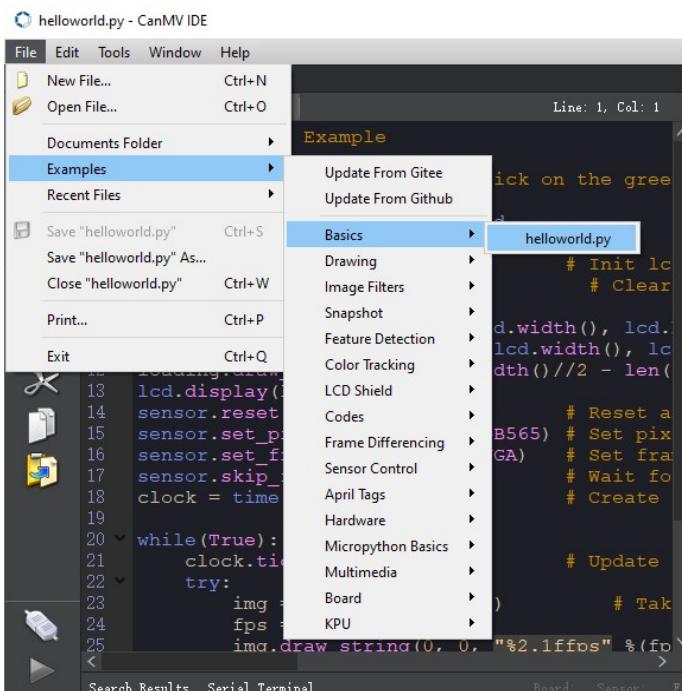
Примечание: при временном запуске программа не сохраняется.

После отключения питания, нажатия кнопки Reset или разрыва соединения программа будет потеряна.

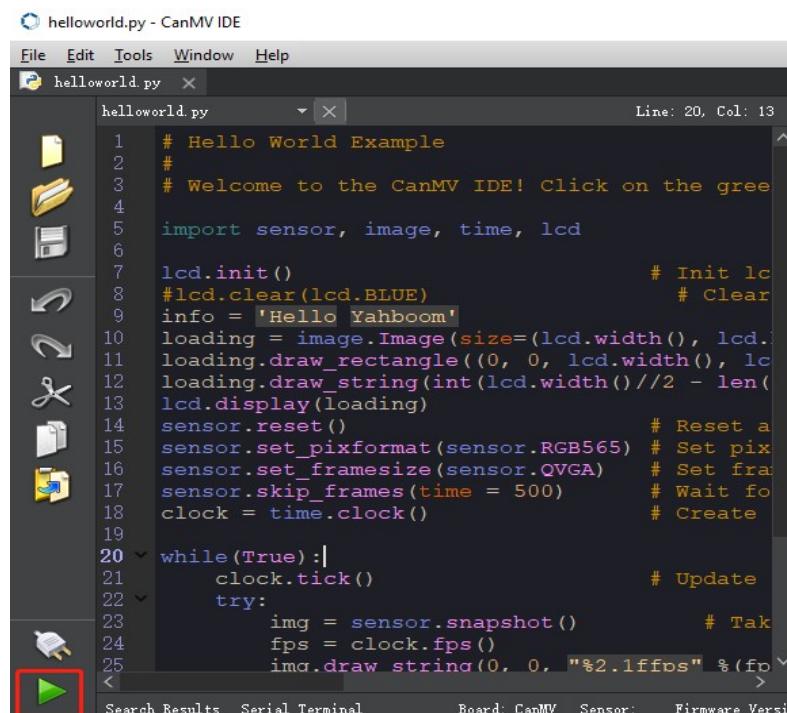
После успешного подключения в левом нижнем углу отображается активный значок соединения.

Если ни один файл не открыт, можно загрузить пример:

File → Examples → Basics → helloworld.py



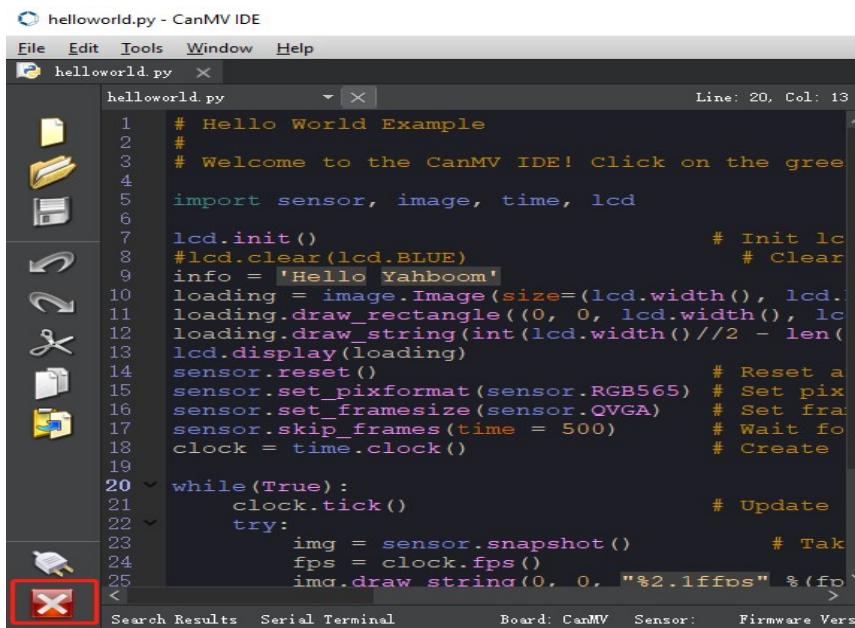
Нажмите зелёную кнопку Play для запуска программы.



В результате:

на LCD-экране модуля отобразится изображение с камеры;
в правом верхнем углу IDE также появится окно предпросмотра камеры.
После запуска:

кнопка Play изменится на красную кнопку Stop;
нажатие Stop завершает выполнение программы.

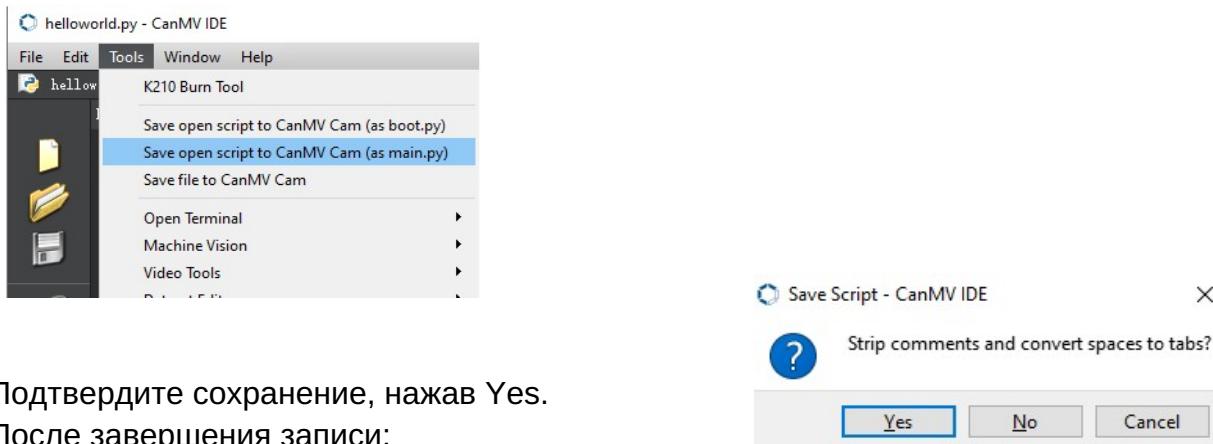


2.5 Автозапуск программы при включении питания (Boot Run)

Примечание: в этом режиме программа автоматически запускается после повторного включения питания.

При активном соединении выполните команду:

Tools → Save the currently open script of main.py to CanMV Cam



Подтвердите сохранение, нажав Yes.

После завершения записи:

можно отключить устройство или нажать кнопку Reset;
программа будет запускаться автоматически при подаче питания.

Если в модуль вставлена TF-карта:
программа сохраняется на TF-карту;
приоритет выполнения отдается файлу main.py, расположенному на TF-карте.

3. Прошивка заводской прошивки

3.1 Описание заводской прошивки

Официальная bare-metal версия SDK для K210 не поддерживает MicroPython, поэтому перед использованием MicroPython необходимо загрузить прошивку с поддержкой CanMV.

Заводская прошивка модуля K210 версии Yahboom основана на официальном проекте CanMV от Kendryte и была доработана с учётом поддержки периферийных устройств модуля.

Модуль K210 с завода уже прошит заводской прошивкой, поэтому после подключения к CanMV IDE можно сразу использовать программирование на MicroPython.

Не требуется перепрошивать заводскую прошивку каждый раз.

Повторная прошивка необходима только в случае:

обновления прошивки;
записи другой (сторонней) прошивки.

Заводская прошивка предоставляется только в виде бинарного файла (.bin),
исходный код не поставляется.

Если вы хотите изучить исходный код базовой прошивки MicroPython для K210,
воспользуйтесь официальным проектом CanMV:

<https://github.com/kendryte/canmv>

3.2 Очистка памяти чипа (Empty Chip Firmware)

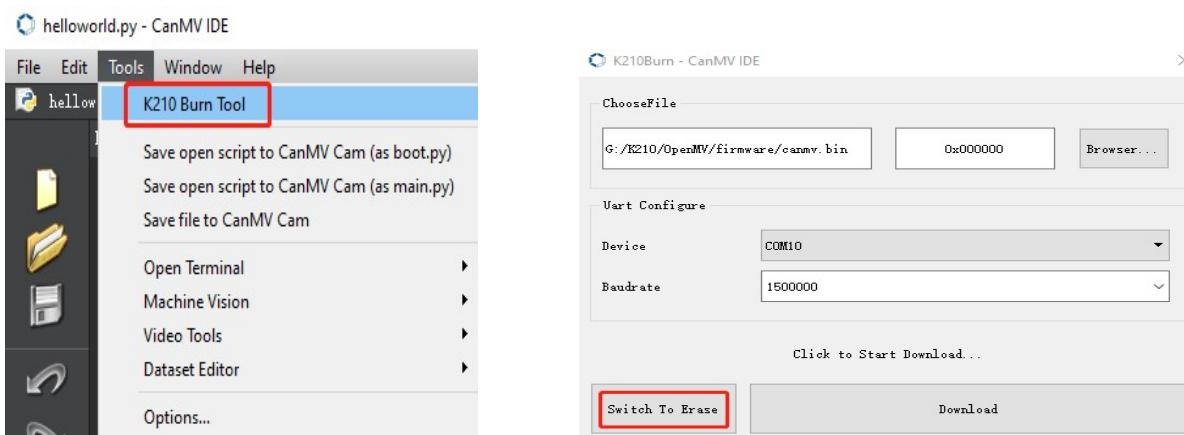
Для восстановления заводской прошивки перед прошивкой необходимо выполнить полную очистку Flash-памяти микроконтроллера.

Внимание:

Данный шаг полностью очищает программное пространство Flash-памяти модуля K210.
После очистки необходимо обязательно заново прошить прошивку, иначе CanMV IDE
не сможет подключиться к модулю.

В CanMV IDE выполните:

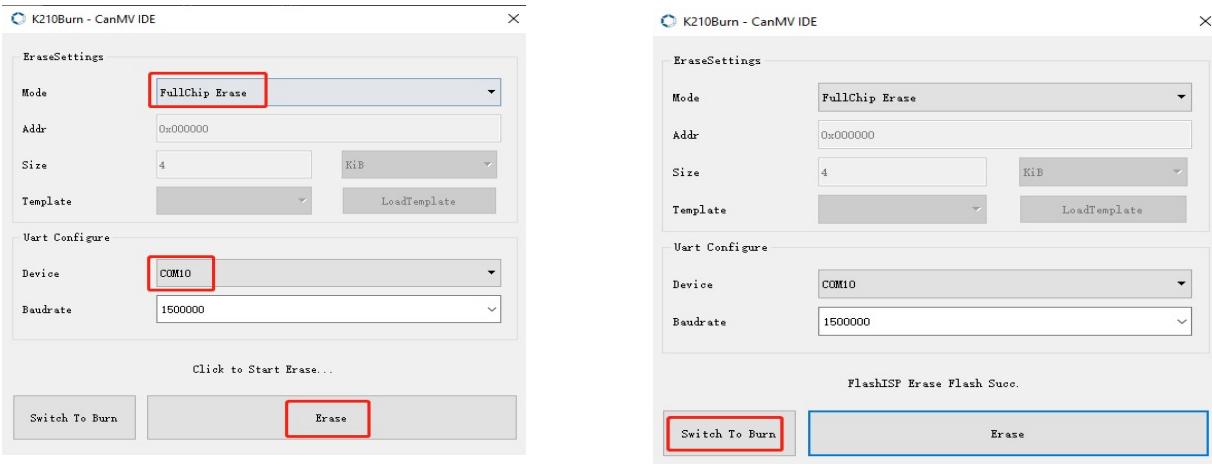
Tools → K210 Burn Tool → Switch To Erase



В настройках Erase:

выберите режим FullChip Erase;
выберите последовательный порт модуля K210;
нажмите EraseStart для начала очистки памяти.

Дождитесь завершения операции очистки, затем переключитесь в режим прошивки для повторной записи прошивки.

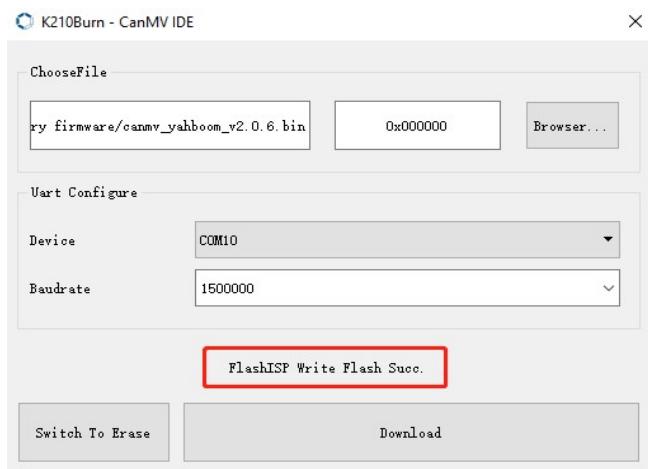
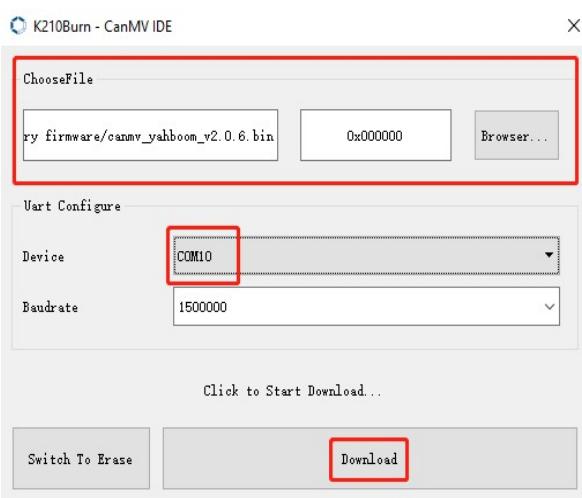


3.3 Прошивка заводской прошивки

В каталоге исходных файлов найдите файл заводской прошивки:

canmv_yahboom_vX.X.X.bin

canmv_yahboom_v2.0.6.bin



где vX.X.X — номер версии прошивки
(в примере используется версия v2.0.6).

В окне прошивки:

нажмите Browse / Select и выберите файл прошивки;
адрес прошивки не изменяйте, используйте значение по умолчанию 0x000000;
в поле Device выберите последовательный порт модуля K210;
нажмите Download для начала прошивки.

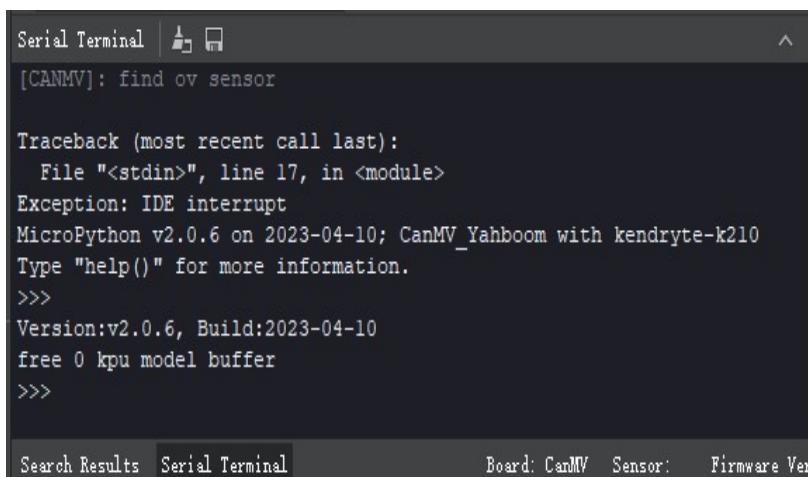
После успешного завершения появится сообщение:

FlashISP Write Flash Succ

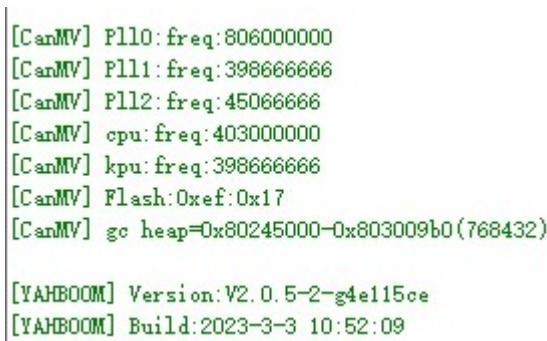
3.4 Просмотр версии прошивки

Подключите модуль K210 к CanMV IDE и откройте встроенный Serial Terminal в нижней части IDE.

При каждом завершении выполнения программы в терминале выводится текущая версия прошивки.

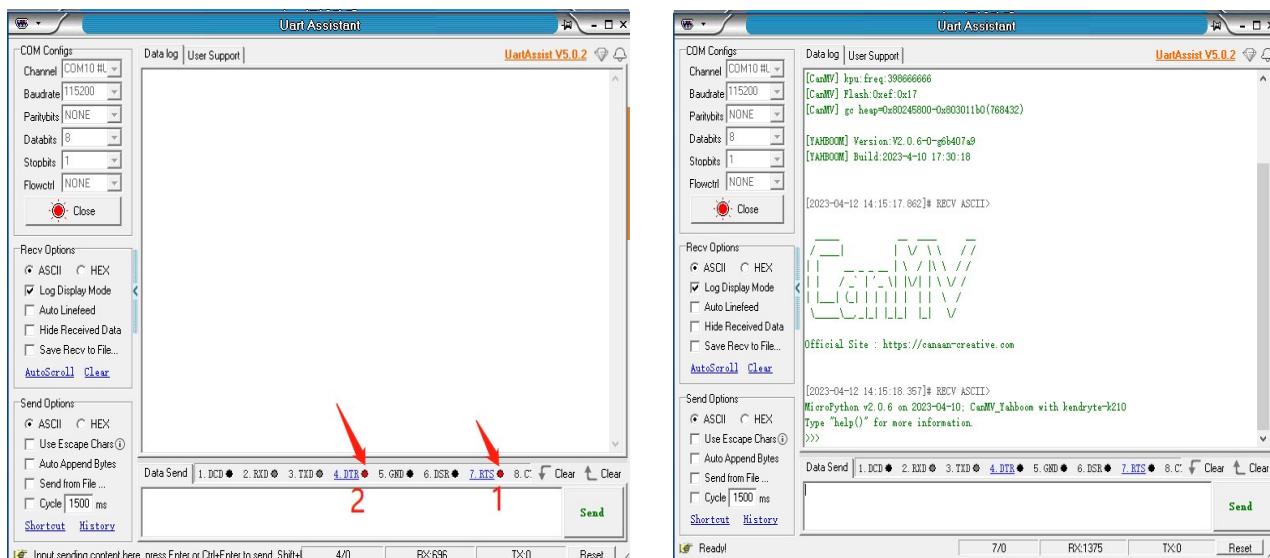


Также можно подключить модуль K210 к последовательному терминалу (например, UartAssist) и просмотреть выводимую информацию там.



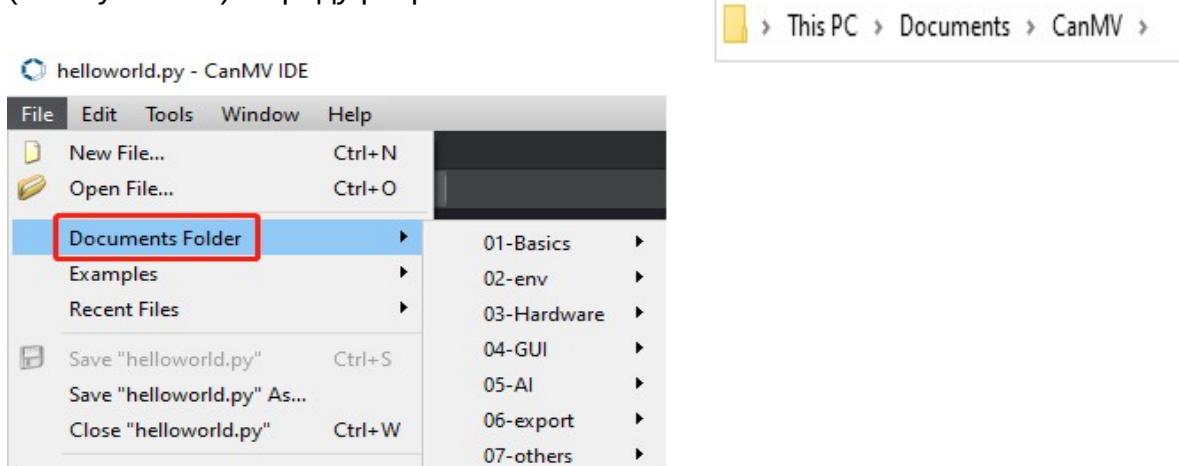
Если используется UartAssist:

после открытия последовательного порта программа не запускается автоматически; необходимо сначала нажать 7.RTS, затем 4.DTR;
после переключения индикаторов из красного состояния в чёрное программа начнёт выполняться корректно.



4. Импорт примеров (рутин) в IDE CanMV

В данном разделе рассматривается процесс загрузки и импорта заводских примеров (factory routine) в среду разработки CanMV IDE.



5. Вызов пользовательской библиотеки Python

5.1 Описание пользовательской библиотеки Python

Для тех, кто уже работал с Python: язык Python позволяет вызывать пользовательские библиотеки .py (а также сторонние библиотеки), расположенные в том же каталоге. Аналогичным образом MicroPython также поддерживает вызов пользовательских библиотек Python.

Перед тем как использовать пользовательскую библиотеку .py, необходимо импортировать файл библиотеки в модуль K210.

5.2 Импорт библиотеки Python

Рассмотрим пример импорта файла simplePID.py.

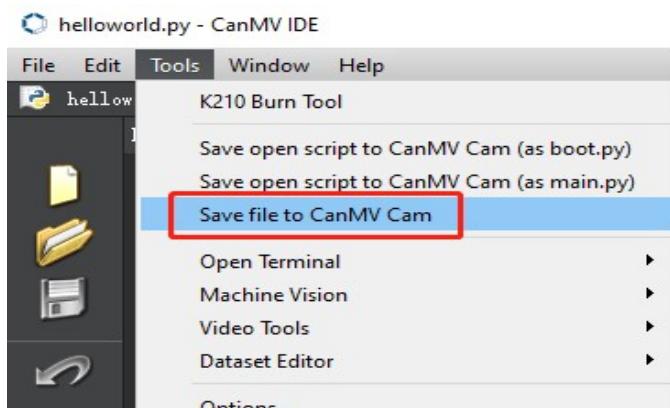
Существует два способа импортировать пользовательскую библиотеку Python.

Первый способ — скопировать файл пользовательской библиотеки .py непосредственно в корневой каталог карты памяти, после чего вставить карту памяти в устройство и использовать библиотеку.

Второй способ — импортировать библиотеку через CanMV IDE.

Подробная последовательность действий описана ниже.

В меню IDE перейдите в Tools → Save file to CanMV Cam.

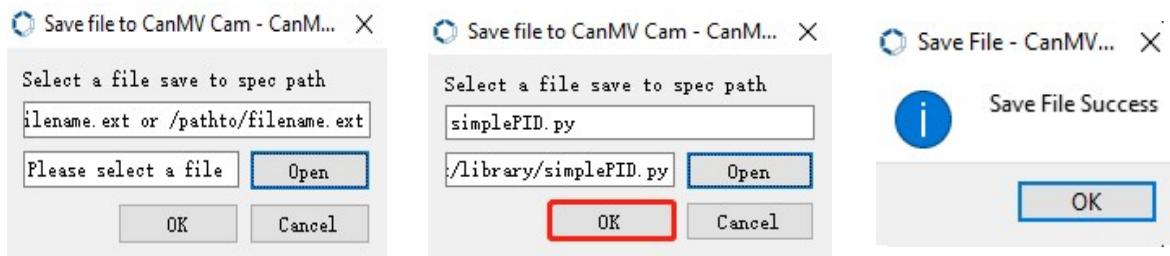


Нажмите Open, выберите файл .py, который необходимо передать.

Измените имя файла на simplePID.py.

Нажмите OK, чтобы начать запись файла.

Примечание: файлы, записываемые данным способом, сначала сохраняются на карту памяти.



5.3 Вызов библиотеки Python

```
from simplePID import PID
print("import simplePID OK")
PID_controller = PID(0, 10, 0, 1)
print("PID test OK")
```

Запустите приведённый выше код. Если всё выполнено корректно, в последовательном терминале, расположенному в нижней части окна IDE, будет выведено сообщение test OK.

```
>>> free 0 kpu model buffer
>>>

>>> free 0 kpu model buffer
>>>

import simplePID OK
PID test OK
MicroPython v2.0.5 on 2023-02-23; CanMV_Yahboom with kendryte-k210
Type "help()" for more information.
>>>
```

Если файл simplePID.py не был импортирован ранее или имя файла/модуля не совпадает, при выполнении программы будет выведено сообщение об ошибке.

Таким образом, успешный вывод сообщений подтверждает корректный импорт и вызов пользовательской библиотеки Python в среде MicroPython на базе K210.

6 Документация по API MicroPython

4) Практический ориентир: где искать нужное API

1) Официальная документация MicroPython (API Reference)

Главная страница документации (включая Library Reference / справочник модулей).

<https://docs.micropython.org/en/latest/>

Раздел “MicroPython language and implementation” (про особенности языка/REPL/реализацию).

<https://docs.micropython.org/en/latest/reference/index.html>

PDF-версия документации (удобно скачать и искать по тексту).

<https://docs.micropython.org/en/latest/micropython-docs.pdf>

Пример страницы конкретного модуля (как устроены все API-страницы):

uos / “os” (файловая система, uname, urandom и т.д.).

<https://docs.micropython.org/en/latest/library/os.html>

2) Если ты на K210 с CanMV: список модулей MicroPython внутри CanMV

У CanMV есть своя страница “Micropython Library” (какие модули есть в прошивке и что из MicroPython-специфичного поддерживается).

Полезная подсказка из CanMV-доков: в терминале можно выполнить `help('modules')`, чтобы увидеть, какие модули реально встроены в твою прошивку.

<https://www.kendryte.com/canmv/main/canmv/library/micropython/README.html>

3) CanMV API (K210): расширения поверх стандартного MicroPython

Для K210/CanMV важны “расширенные” модули (камера/изображения/экран/maix и т.п.).

Индекс CanMV API (K210): список расширенных модулей вроде sensor, image, lcd, maix.* и др.

https://kendryte.github.io/canmv_docs/main/canmv/library/index.html

Репозиторий CanMV (если нужно сверяться с исходниками/примерами).

<https://github.com/kendryte/canmv>

4) Практический ориентир: где искать нужное API

“Стандартное” MicroPython: machine, uos, time/utime, sys, socket и т.д. — смотри в официальном Library Reference.

<https://docs.micropython.org/en/latest/>

“Железо и комп. зрение на K210 (CanMV)”: sensor, image, lcd, maix.* — смотри в CanMV API manual.

https://kendryte.github.io/canmv_docs/main/canmv/library/index.html

Руководство по API CanMV

https://kendryte.github.io/canmv_docs/main/canmv/library/index.html

4. Процедуры графических манипуляций

1. Рисование графики (Draw graphics)

1.1 Цель эксперимента

В данном эксперименте вы познакомитесь с основами рисования графических примитивов в CanMV (MicroPython) для K210.

В ходе эксперимента вы научитесь:

выводить текст на LCD-дисплей;

рисовать линию;

рисовать прямоугольник (с заливкой и без);

рисовать окружность;

работать с цветами в формате RGB.

Файл примера из документации:

CanMV -> 04-GUI -> geometric.py

1.2 Ход эксперимента

Важно перед началом

Заводская прошивка модуля K210 уже содержит встроенные модули lcd и image.

Если ранее вы устанавливали другую прошивку, необходимо перепрошить плату заводской прошивкой, иначе данный пример работать не будет.

Шаг 1. Импорт необходимых модулей

```
import lcd, image
```

lcd — отвечает за работу с LCD-дисплеем;

image — используется для создания изображения и рисования графики.

Шаг 2. Инициализация LCD и создание изображения

```
lcd.init()  
img = image.Image(size=(lcd.width(), lcd.height()))
```

Пояснение:

lcd.init() — инициализирует LCD-дисплей;

image.Image(...) — создаёт изображение (буфер) размером, равным разрешению экрана.

Важно понимать:

рисование происходит не напрямую на LCD, а в объекте img. Только после этого изображение выводится на экран.

Шаг 3. Заливка фона цветом

```
img.draw_rectangle(  
    (0, 0, lcd.width(), lcd.height()),  
    fill=True,  
    color=(0, 81, 137)  
)
```

Пояснение:

рисуется прямоугольник на весь экран;

fill=True — включает заливку;

цвет задаётся в формате RGB (R, G, B).

Шаг 4. Рисование текста и графических примитивов

Вывод текста

```
img.draw_string(  
    50, 10,  
    "Hello Yahboom",  
    color=(255, 255, 255),  
    scale=2  
)
```

(50, 10) — координаты начала текста;
scale=2 — масштаб текста (увеличение в 2 раза).

Рисование линии

```
img.draw_line(  
    50, 50,  
    200, 50,  
    color=(100, 0, 0),  
    thickness=5  
)
```

линия от точки (50, 50) до (200, 50);
thickness — толщина линии в пикселях.

Рисование прямоугольника

```
img.draw_rectangle(  
    50, 60,  
    150, 150,  
    color=(0, 100, 0),  
    thickness=2,  
    fill=True  
)
```

первые два параметра — координаты;
следующие два — ширина и высота;
fill=True — прямоугольник закрашен.

Рисование окружности

```
img.draw_circle(  
    125, 135,  
    50,  
    color=(255, 255, 255),  
    thickness=2,  
    fill=False  
)
```

(125, 135) — центр окружности;
50 — радиус;
fill=False — рисуется только контур.

Шаг 5. Вывод изображения на LCD

```
lcd.display(img)
```

Без этой команды изображение не появится на экране.

1.3 Результаты эксперимента

Подключите модуль K210 к компьютеру с помощью microUSB-кабеля.

В CanMV IDE нажмите кнопку Connect.

После подключения нажмите Run для запуска программы.

Также вы можете сохранить файл как main.py — тогда программа будет автоматически запускаться при включении платы.

Ожидаемый результат

На LCD-дисплее отобразятся:

надпись "Hello Yahboom";

горизонтальная линия;

зелёный прямоугольник;

белая окружность.

1.4 Итоги эксперимента

Используя CanMV IDE и MicroPython, можно быстро и удобно создавать простую графику для LCD-дисплея K210.

Этот подход является основой для:

GUI-интерфейсов;

отладочных экранов;

визуализации данных;

дальнейших проектов с камерой и AI.

Дополнительные пояснения и примеры

Пример 1. Прямоугольник без заливки

```
img.draw_rectangle(20, 20, 100, 60, color=(255, 0, 0), fill=False)
```

Используется для рамок и интерфейсных элементов.

Пример 2. Закрашенная окружность (кнопка)

```
img.draw_circle(160, 120, 20, color=(0, 0, 255), fill=True)
```

Можно использовать как кнопку или индикатор состояния.

Пример 3. Очистка экрана другим цветом

```
img.clear()  
img.draw_rectangle((0, 0, lcd.width(), lcd.height()), fill=True, color=(0, 0, 0))  
lcd.display(img)
```

Важное замечание

Для анимаций и интерфейсов обычно используют:

цикл while True;

перерисовку img;

повторный вызов lcd.display(img).

```
# Импортируем модули CanMV:  
# lcd — управление LCD-дисплеем (инициализация, вывод изображения на экран)  
# image — создание изображения (буфера) и рисование графики (текст, линии, фигуры)  
import lcd, image  
  
# Инициализация LCD-дисплея.  
# После этой команды CanMV знает размеры экрана и готов выводить изображение.  
lcd.init()  
  
# Создаём объект изображения (буфер кадра) размером ровно как экран LCD:  
# lcd.width() — ширина экрана в пикселях  
# lcd.height() — высота экрана в пикселях  
#  
# Важно: мы рисуем НЕ "на экран", а в картинку img.  
# Экран обновится только после lcd.display(img).  
img = image.Image(size=(lcd.width(), lcd.height()))  
  
# Заливаем фон: рисуем прямоугольник на весь экран и включаем заливку fill=True.  
# Координаты прямоугольника задаются как (x, y, w, h):  
# x, y — левый верхний угол  
# w, h — ширина и высота  
#  
# color=(R, G, B) — цвет в формате RGB, значения 0..255  
# (0, 81, 137) — тёмно-синий/бирюзовый фон  
img.draw_rectangle((0, 0, lcd.width(), lcd.height()), fill=True, color=(0, 81, 137))  
  
# Пишем текст на изображении:  
# (50, 10) — позиция начала строки (левый верхний угол текста)  
# color — белый (255,255,255)  
# scale=2 — увеличить текст в 2 раза  
img.draw_string(50, 10, "Hello Yahboom", color=(255, 255, 255), scale=2)  
  
# Рисуем линию:  
# img.draw_line(x1, y1, x2, y2, ...)  
# Толщина линии задаётся параметром thickness (в пикселях).  
# Цвет (100,0,0) — тёмно-красный.  
img.draw_line(50, 50, 200, 50, color=(100, 0, 0), thickness=5)  
  
# Рисуем прямоугольник (закрашенный):  
# Здесь используется другая форма записи: (x, y, w, h) как отдельные параметры.  
# (50, 60) — левый верхний угол  
# 150, 150 — ширина и высота  
# thickness=2 — толщина рамки (контура)  
# fill=True — закрашиваем внутреннюю область  
# color=(0,100,0) — зелёный  
img.draw_rectangle(50, 60, 150, 150, color=(0, 100, 0), thickness=2, fill=True)  
  
# Рисуем окружность:  
# img.draw_circle(cx, cy, r, ...)  
# cx, cy — центр окружности  
# r — радиус  
# fill=False — только контур (без заливки)  
# thickness=2 — толщина контура  
# color=(255,255,255) — белый  
img.draw_circle(125, 135, 50, color=(255, 255, 255), thickness=2, fill=False)  
  
# Выводим итоговое изображение img на LCD.  
# Без этой команды ничего не появится на экране, потому что рисование шло в буфер img.  
lcd.display(img)
```

2. Кнопка и событие нажатия

2.1 Цель эксперимента

В данном эксперименте вы научитесь:

создавать кнопки с использованием графической библиотеки lvgl;
обрабатывать событие нажатия кнопки (click event).

Эталонный путь коду для данного эксперимента:

CanMV -> 04-GUI -> button.py

2.2 Ход эксперимента

Заводская прошивка модуля уже содержит интегрированную графическую библиотеку lvgl.
Если ранее была прошита другая прошивка, перед выполнением эксперимента необходимо заново записать заводскую прошивку.

1. Импорт необходимых библиотек

```
import lvgl as lv
import lvgl_helper as lv_h
import lcd
import time
from machine import Timer
import touchscreen as ts
```

2. Инициализация LCD, сенсорного экрана и lvgl

```
lcd.init()
ts.init()
lv.init()
```

3. Подключение драйвера дисплея K210 к интерфейсу регистрации дисплея lvgl

```
disp_buf1 = lv.disp_buf_t()
buf1_1 = bytearray(320*10)
lv.disp_buf_init(disp_buf1, buf1_1, None, len(buf1_1)//4)

disp_drv = lv.disp_drv_t()
lv.disp_drv_init(disp_drv)
disp_drv.buffer = disp_buf1
disp_drv.flush_cb = lv_h.flush
disp_drv.hor_res = 320
disp_drv.ver_res = 240
lv.disp_drv_register(disp_drv)
```

4. Подключение драйвера сенсорного экрана K210 к интерфейсу ввода lvgl

```
indev_drv = lv.indev_drv_t()
lv.indev_drv_init(indev_drv)
indev_drv.type = lv.INDEV_TYPE.POINTER
indev_drv.read_cb = lv_h.read
lv.indev_drv_register(indev_drv)
```

5. Создание кнопки и отображение надписи «Button»

```
scr = lv.obj()
btn = lv.btn(scr)
btn.align(lv.scr_act(), lv.ALIGN.CENTER, 0, 0)

label = lv.label(btn)
label.set_text("Button")
label.set_size(20,20)

lv.scr_load(scr)
btn.set_event_cb(on_btn_cb)
```

6. Обработка события нажатия кнопки

Каждый раз при нажатии кнопки значение переменной `btn_count` увеличивается на 1, и новое значение отображается на кнопке.

```
btn_count = 0
```

```
def on_btn_cb(obj, event):
    global btn_count
    if event == lv.EVENT.CLICKED:
        btn_count += 1
        label.set_text(str(btn_count))
        print("Button Press:", btn_count)
```

7. Периодическое обновление задач lvgl

Так как интерфейс lvgl требует постоянного обновления, необходимо вызывать обработчик задач каждые 5 мс.

```
tim = time.ticks_ms()
while True:
    if time.ticks_ms() - tim > 5:
        tim = time.ticks_ms()
        lv.task_handler()
        lv.tick_inc(5)
```

2.3 Результаты эксперимента

Подключите модуль K210 к компьютеру с помощью кабеля microUSB.

В среде CanMV IDE нажмите кнопку Connect, после успешного подключения — кнопку Run для запуска программы.

Также можно сохранить код как `main.py`, загрузить его в модуль K210 и запускать автономно.

На экране LCD появится синяя кнопка с надписью “Button”, расположенная по центру. Каждое касание кнопки увеличивает число на кнопке на 1.



При повторном запуске программы рекомендуется переподключить устройство, чтобы избежать наложения числовых значений на кнопке.

2.4 Итоги эксперимента

Использование CanMV IDE и синтаксиса MicroPython в заводской прошивке позволяет очень удобно создавать графический интерфейс и обрабатывать события касания кнопок.

В реальных проектах обработку событий кнопок рекомендуется выполнять без вывода данных через `print`, однако в данном эксперименте вывод в терминал используется исключительно в учебных целях.

```

# -----
# GUI на K210 (CanMV) + LVGL
# Тема: Кнопка и обработка клика
# -----

# LVGL — графическая библиотека (объекты, события, отрисовка интерфейса)
import lvgl as lv

# lvgl_helper — «мост» между LVGL и драйверами железа CanMV:
# - flush: вывод буфера на экран
# - read: чтение координат касания
import lvgl_helper as lv_h

# lcd — драйвер экрана (инициализация и низкоуровневый вывод)
import lcd

# time — нужен для тайминга (периодический вызов task_handler)
import time

# Timer здесь не используется в данном примере,
# но часто нужен для аппаратного таймера вместо бесконечного цикла
from machine import Timer

# touchscreen — драйвер тачскрина (инициализация сенсорной панели)
import touchscreen as ts

# --- 1) Инициализация оборудования и LVGL ---

# Инициализируем дисплей (подготовка контроллера и параметров вывода)
lcd.init()

# Инициализируем сенсорную панель (подготовка чтения касаний)
ts.init()

# Инициализируем LVGL (внутренние структуры, память, обработчики)
lv.init()

# --- 2) Регистрация дисплея в LVGL ---

# LVGL рисует всё в «буфер кадра» (частичный буфер).
# Затем драйвер по callback-функции отправляет этот буфер на реальный экран.
disp_buf1 = lv.disp_buf_t()

# Создаём небольшой буфер в памяти:
# 320*10 означает: 320 пикселей по ширине и 10 строк по высоте.
# Это не полный экран, а «полоса», которую LVGL будет обновлять частями.
buf1_1 = bytearray(320 * 10)

# Инициализируем буфер дисплея:
# - disp_buf1: структура LVGL для буфера
# - buf1_1: реальный массив байт
# - None: второй буфер не используем (одинарная буферизация)
# - len(buf1_1)//4: размер в «пикселях/словах» (зависит от конфигурации цвета)
lv.disp_buf_init(disp_buf1, buf1_1, None, len(buf1_1) // 4)

# Создаём драйвер дисплея LVGL и заполняем его параметрами
disp_drv = lv.disp_drv_t()
lv.disp_drv_init(disp_drv)

# Подключаем наш буфер к драйверу дисплея
disp_drv.buffer = disp_buf1

# flush_cb — функция, которую LVGL вызывает, когда нужно «слить» буфер на экран
disp_drv.flush_cb = lv_h.flush

# Разрешение экрана (важно указать правильные размеры)
disp_drv.hor_res = 320
disp_drv.ver_res = 240

# Регистрируем дисплей в LVGL (после этого LVGL «знает», куда рисовать)
lv.disp_drv_register(disp_drv)

```

```

# --- 3) Регистрация устройства ввода (тачскрин) в LVGL ---
# LVGL поддерживает разные типы ввода: мышь/тач/клавиатура/энкодер.
# Здесь мы подключаем «указатель» (POINTER) — это тачскрин.
indev_drv = lv.indev_drv_t()
lv.indev_drv_init(indev_drv)

# Тип устройства ввода — POINTER (координаты касания как у мыши)
indev_drv.type = lv.INDEV_TYPE.POINTER

# read_cb — функция, которую LVGL вызывает для чтения текущего касания
indev_drv.read_cb = lv_h.read

# Регистрируем устройство ввода в LVGL
lv.indev_drv_register(indev_drv)

# --- 4) Обработчик события кнопки (клик) ---
# Счётчик нажатий — будем увеличивать при каждом клике
btn_count = 0

def on_btn_cb(obj, event):
    """
    Callback-функция LVGL: вызывается при событиях на объекте (кнопке).

    obj — объект, который сгенерировал событие (наша кнопка)
    event — тип события (CLICKED, PRESSED, RELEASED и т.д.)
    """
    global btn_count

    # Нас интересует именно «клик» (касание + отпускание)
    if event == lv.EVENT.CLICKED:
        btn_count += 1

    # Меняем текст на метке (label) внутри кнопки
    label.set_text(str(btn_count))

    # Печать в консоль удобна для обучения,
    # но в реальном GUI может тормозить обработку событий
    print("Button Press:", btn_count)

# --- 5) Создание интерфейса (экран, кнопка, надпись) ---
# Создаём новый объект-экран (контейнер верхнего уровня)
scr = lv.obj()

# Создаём кнопку на этом экране
btn = lv.btn(scr)

# Выравниваем кнопку по центру активного экрана
# (x=0, y=0 — без смещения)
btn.align(lv.scr_act(), lv.ALIGN.CENTER, 0, 0)

# Создаём надпись (label) внутри кнопки
label = lv.label(btn)
label.set_text("Button")

# Размер label в пикселях:
# Важно: set_size — это размер области. Текст может быть больше/меньше.
label.set_size(20, 20)

# Назначаем обработчик событий на кнопку
btn.set_event_cb(on_btn_cb)

# Загружаем наш экран и делаем его активным (показываем на дисплее)
lv.scr_load(scr)

# --- 6) Главный цикл LVGL: обработка задач и «тик» времени ---
# LVGL — событийная система, но ей нужно регулярно:
# - обрабатывать задачи/анимации/перерисовки (lv.task_handler)
# - получать «тиковое» время (lv.tick_inc)
#
# Здесь мы делаем это вручную каждые 5 мс.

tim = time.ticks_ms()
while True:
    # Проверяем прошло ли больше 5 мс
    if time.ticks_ms() - tim > 5:
        tim = time.ticks_ms()

    # Обработка внутренних задач LVGL
    lv.task_handler()

    # Сообщаем LVGL, что прошло 5 мс (для таймеров/анимаций)
    lv.tick_inc(5)

```

3. Слайдер (Slider)

3.1 Цель эксперимента

В данном эксперименте вы изучите, как создать и использовать элемент управления Slider (ползунок) с помощью графической библиотеки LVGL на платформе K210.

Путь к эталонному примеру:

CanMV -> 04-GUI -> slider.py

1) Примечания по прошивке

Заводская прошивка, используемая в данном эксперименте, уже содержит интегрированную графическую библиотеку LVGL.

Если ранее была прошита другая прошивка, перед запуском примера необходимо повторно прошить заводскую прошивку.

2) Импорт библиотек

```
import lvgl as lv
import lvgl_helper as lv_h
import lcd
import time
import touchscreen as ts
```

3) Инициализация LCD, сенсорного экрана и LVGL

```
lcd.init()
ts.init()
lv.init()
```

4) Регистрация драйвера дисплея в LVGL

```
disp_buf1 = lv.disp_buf_t()
buf1_1 = bytearray(320 * 10)
```

```
lv.disp_buf_init(disp_buf1, buf1_1, None, len(buf1_1) // 4)
```

```
disp_drv = lv.disp_drv_t()
lv.disp_drv_init(disp_drv)
```

```
disp_drv.buffer = disp_buf1
disp_drv.flush_cb = lv_h.flush
disp_drv.hor_res = 320
disp_drv.ver_res = 240
```

```
lv.disp_drv_register(disp_drv)
```

На данном этапе:

создаётся буфер отображения;
настраивается драйвер дисплея;
драйвер регистрируется в системе LVGL.

5) Регистрация драйвера сенсорного экрана в LVGL

```
indev_drv = lv.indev_drv_t()
lv.indev_drv_init(indev_drv)

indev_drv.type = lv.INDEV_TYPE.POINTER
indev_drv.read_cb = lv_h.read

lv.indev_drv_register(indev_drv)
```

6) Создание слайдера и текстовой метки, обработка событий

```
import lvgl as lv
import lvgl_helper as lv_h
import lcd
import time
import touchscreen as ts

lcd.init()
ts.init()
lv.init()

# Регистрация дисплея
disp_buf1 = lv.disp_buf_t()
buf1_1 = bytearray(320 * 10)
lv.disp_buf_init(disp_buf1, buf1_1, None, len(buf1_1) // 4)

disp_drv = lv.disp_drv_t()
lv.disp_drv_init(disp_drv)
disp_drv.buffer = disp_buf1
disp_drv.flush_cb = lv_h.flush
disp_drv.hor_res = 320
disp_drv.ver_res = 240
lv.disp_drv_register(disp_drv)

# Регистрация сенсора
indev_drv = lv.indev_drv_t()
lv.indev_drv_init(indev_drv)
indev_drv.type = lv.INDEV_TYPE.POINTER
indev_drv.read_cb = lv_h.read
lv.indev_drv_register(indev_drv)

# Создание экрана
scr = lv.obj()

# Создание текстовой метки
textView = lv.label(scr)
textView.align(lv.scr_act(), lv.ALIGN.CENTER, -50, -50)
textView.set_text("Value: 0")

# Создание слайдера
slider = lv.slider(scr)
slider.align(lv.scr_act(), lv.ALIGN.CENTER, 0, 0)
slider.set_width(200)
slider.set_height(30)
slider.set_range(0, 100)
slider.set_value(0, 0)

# Обработчик события изменения значения
def on_slider_changed(obj, event):
    if event == lv.EVENT.VALUE_CHANGED:
        slider_value = slider.get_value()
        textView.set_text("Value: %d" % slider_value)
        print("slider:", slider_value)

slider.set_event_cb(on_slider_changed)

lv.scr_load(scr)

# Основной цикл обработки LVGL
tim = time.ticks_ms()
while True:
    if time.ticks_ms() - tim > 5:
        tim = time.ticks_ms()
        lv.task_handler()
        lv.tick_inc(5)
```

Назначение обработчика событий:

При касании или перемещении ползунка текущее значение слайдера считывается и в реальном времени отображается в текстовой метке.

Примечание: в реальных проектах вывод в консоль (print) внутри обработчика событий может снижать отзывчивость интерфейса. В данном примере он используется исключительно для обучения и отладки.

3.3 Результаты эксперимента

Подключите плату K210 к компьютеру через microUSB

В CanMV IDE нажмите Connect

Нажмите Run

(или сохраните файл как main.py и выполните загрузку с модуля)

В результате на экране вы увидите:

ползунок, расположенный по центру LCD;

текстовую метку над ним;

при перемещении ползунка значение в метке будет изменяться, например:

Value: 37, в реальном времени.



3.4 Вывод по эксперименту

Используя CanMV IDE, MicroPython и заводскую прошивку K210, можно быстро:

создать элемент управления Slider в LVGL;

зарегистрировать обработчик изменения значения;

динамически обновлять элементы интерфейса при взаимодействии пользователя.

При необходимости можно дополнительно реализовать:

шаг изменения значения (например, с шагом 5);

отображение значения непосредственно на бегунке слайдера;

настройку внешнего вида (цвета, стили, темы LVGL).

```

# --- Импорт библиотек ---
import lvgl as lv      # Основная библиотека LVGL (виджеты, события, экраны)
import lvgl_helper as lv_h # Вспомогательные функции CanMV для LVGL (flush/read для дисплея и тача)
import lcd            # Работа с LCD-дисплеем (инициализация, параметры)
import time           # Таймеры/задержки, ticks_ms для системного времени
from machine import Timer # Аппаратные/программные таймеры MicroPython (в этом примере не используется)
import touchscreen as ts # Драйвер сенсорного экрана (инициализация тач-панели)

# --- Инициализация железа и LVGL ---
lcd.init()    # Запускаем LCD (без этого LVGL не сможет выводить графику)
ts.init()     # Запускаем сенсорный экран (чтобы считывать касания)
lv.init()     # Инициализируем LVGL (создание объектов/экранов, обработка задач)

# --- Регистрация дисплея в LVGL ---
disp_buf1 = lv.disp_buf_t()      # Структура LVGL для буфера отображения (double-buffer/one-buffer режим)
buf1_1 = bytearray(320 * 10)    # Память под буфер строк: 320 пикселей * 10 строк (частичный буфер)
# disp_buf_init: (структура буфера, buf1, buf2, размер_в_пикселях)
# Важно: LVGL ожидает размер буфера в "кол-ве пикселей", поэтому часто делят на 4 (зависит от формата цвета/памяти в порте)
lv.disp_buf_init(disp_buf1, buf1_1, None, len(buf1_1) // 4)

disp_drv = lv.disp_drv_t()       # Структура драйвера дисплея LVGL
lv.disp_drv_init(disp_drv)      # Заполняем структуру значениями по умолчанию

disp_drv.buffer = disp_buf1     # Подключаем наш буфер отрисовки
disp_drv.flush_cb = lv_h.flush # Колбэк "flush": LVGL вызывает его, когда надо вывести готовые пиксели на LCD
disp_drv.hor_res = 320          # Горизонтальное разрешение экрана
disp_drv.ver_res = 240          # Вертикальное разрешение экрана
lv.disp_drv_register(disp_drv) # Регистрируем драйвер дисплея в LVGL

# --- Регистрация сенсора (тача) в LVGL ---
indev_drv = lv.indev_drv_t()    # Структура драйвера устройства ввода LVGL
lv.indev_drv_init(indev_drv)    # Значения по умолчанию

indev_drv.type = lv.INDEV_TYPE.POINTER # Тип ввода: POINTER (курсор/палец) — нужен для тачскрина
indev_drv.read_cb = lv_h.read       # Колбэк чтения: LVGL вызывает его, чтобы получить координаты касания
lv.indev_drv_register(indev_drv)   # Регистрируем устройство ввода

# --- Обработчик события слайдера ---
# В LVGL колбэк обычно имеет вид: callback(obj, event)
# Здесь в сигнатуре есть лишние параметры self/obj по шаблону, но реально используется глобальный slider/textView.
def on_slider_changed(self, obj=None, event=-1):
    # Читаем текущее значение слайдера (0..100)
    slider_value = slider.get_value()

    # Обновляем текст метки: показываем текущее значение
    textView.set_text("Value: %d" % (slider_value))

    # Выводим в терминал (полезно для отладки, но может тормозить интерфейс)
    print("slider:", slider_value)

# --- Создание объектов интерфейса ---
scr = lv.obj()                 # Создаём корневой контейнер-экран (screen)

# Слайдер
slider = lv.slider(scr)        # Создаём слайдер и делаем его дочерним объектом экрана scr
slider.align(lv.scr_act(), lv.ALIGN.CENTER, 0, 0) # Выравниваем по центру активного экрана
slider.set_width(200)           # Ширина слайдера
slider.set_height(30)           # Высота слайдера
slider.set_range(0, 100)         # Диапазон значений (min=0, max=100)
slider.set_value(0, 0)           # Устанавливаем стартовое значение 0 (второй параметр: анимация 0=нет)
slider.set_event_cb(on_slider_changed) # Назначаем обработчик событий (будет вызываться при событиях, в т.ч. VALUE_CHANGED)

# Метка (label) для отображения значения
textView = lv.label(scr)       # Создаём label на экране scr
textView.align(lv.scr_act(), lv.ALIGN.CENTER, -50, -50) # Смещаем относительно центра вверх-влево
textView.set_text("Value:0")    # Начальный текст (можно сделать "Value: 0" для единообразия)

lv.scr_load(scr)               # Загружаем наш экран scr как активный (показываем на дисплее)

# --- Главный цикл LVGL ---
# LVGL в MicroPython обычно требует:
# 1) lv.task_handler() — обработка задач (перерисовка, анимации, события)
# 2) lv.tick_inc(ms) — сообщаем LVGL, сколько времени прошло
tim = time.ticks_ms()
while True:
    if time.ticks_ms() - tim > 5: # Каждые ~5 мс
        tim = time.ticks_ms()
        lv.task_handler()          # Обработка задач LVGL
        lv.tick_inc(5)             # Увеличиваем внутренний "таймер" LVGL на 5 мс

```

4. Анимация графика (Chart Animation)

4.1 Цель эксперимента

В данном эксперименте вы научитесь создавать анимированный график (chart) с использованием графической библиотеки LVGL в среде MicroPython на модуле K210.

В ходе эксперимента будет реализована анимация изменения диапазона значений графика, а также управление скоростью анимации с помощью слайдера.

Путь к эталонному примеру кода:

CanMV -> 04-GUI -> chart_anime.py

4.2 Ход эксперимента

Заводская прошивка модуля K210 уже содержит интегрированную графическую библиотеку LVGL.

Если ранее была установлена другая прошивка, перед выполнением эксперимента необходимо перепрошить модуль заводской прошивкой.

1. Инициализация LVGL и устройств ввода/вывода

На первом этапе выполняется:

импорт необходимых библиотек;
инициализация LVGL;
регистрация драйвера дисплея;
регистрация драйвера устройства ввода (тачскрин).

Это обязательный шаг для корректной работы графического интерфейса.

2. Класс анимации Anim

Для удобства работы с анимацией создаётся кастомный класс Anim, наследуемый от lv.anim_t.

```
class Anim(lv.anim_t):
    def __init__(self, obj, val, size, exec_cb, path_cb, time=500, playback=False, ready_cb=None):
        super().__init__()
        lv.anim_init(self)
        lv.anim_set_time(self, time, 0)
        lv.anim_set_values(self, val, val + size)
        if callable(exec_cb):
            lv.anim_set_custom_exec_cb(self, exec_cb)
        else:
            lv.anim_set_exec_cb(self, obj, exec_cb)
        lv.anim_set_path_cb(self, path_cb)
        if playback:
            lv.anim_set_playback(self, 0)
        if ready_cb:
            lv.anim_set_ready_cb(self, ready_cb)
        lv.anim_create(self)
```

Назначение класса:

хранит параметры анимации;
задаёт начальное и конечное значение;
определяет кривую анимации (ease-in / ease-out);

позволяет выполнять пользовательские функции при завершении анимации.

3. Класс анимированного графика AnimatedChart

Создаётся класс AnimatedChart, наследуемый от lv.chart, который управляет логикой анимации графика.

```
class AnimatedChart(lv.chart):
    def __init__(self, parent, val, size):
        super().__init__(parent)
        self.val = val
        self.size = size
        self.max = 2000
        self.min = 500
        self.factor = 100
        self.anim_phase1()
```

Основные параметры:

val — начальное значение диапазона;
size — амплитуда изменения;
factor — коэффициент скорости анимации (изменяется слайдером).

Фаза 1 — увеличение диапазона

```
def anim_phase1(self):
    Anim(
        self,
        self.val,
        self.size,
        lambda a, val: self.set_range(0, val),
        lv.anim_path_ease_in,
        ready_cb=lambda a: self.anim_phase2(),
        time=(self.max * self.factor) // 100)
```

Фаза 2 — уменьшение диапазона

```
def anim_phase2(self):
    Anim(
        self,
        self.val + self.size,
        -self.size,
        lambda a, val: self.set_range(0, val),
        lv.anim_path_ease_out,
        ready_cb=lambda a: self.anim_phase1(),
        time=(self.min * self.factor) // 100)
```

Диапазон плавно уменьшается с кривой ease_out, после чего анимация повторяется циклически.

4. Настройка графика

```
scr = lv.obj()
chart = AnimatedChart(scr, 100, 1000)
chart.set_width(scr.get_width() - 100)
chart.set_height(scr.get_height() - 60)
chart.align(scr, lv.ALIGN.CENTER, 0, 0)
```

график размещается по центру экрана;
размеры подгоняются под дисплей.

Серия данных и внешний вид

```
series1 = chart.add_series(lv.color_hex(0xFF0000))
chart.set_type(chart.TYPE.POINT | chart.TYPE.LINE)
chart.set_series_width(3)
chart.set_range(0, 100)
```

красная линия;
точки + линия;
толщина линии — 3 пикселя.

Данные графика и оси

```
chart.init_points(series1, 10)
chart.set_points(series1, [10,20,35,20,10,40,51,90,95,80])
chart.set_x_tick_texts('a\nb\nc\nd\ne', 2, lv.chart.AXIS.DRAW_LAST_TICK)
chart.set_y_tick_texts('1\n2\n3\n4\n5', 2, lv.chart.AXIS.DRAW_LAST_TICK)
```

5. Слайдер управления скоростью анимации

```
def on_slider_changed(self, obj=None, event=-1):
    chart.factor = slider.get_value()
```

Создаётся вертикальный слайдер, который изменяет коэффициент factor.

```
def on_slider_changed(self, obj=None, event=-1):
    chart.factor = slider.get_value()
```

```
slider = lv.slider(scr)
slider.align(chart, lv.ALIGN.OUT_RIGHT_TOP, 10, 0)
slider.set_width(30)
slider.set_height(chart.get_height())
slider.set_range(10, 200)
slider.set_value(chart.factor, 0)
slider.set_event_cb(on_slider_changed)
```

Чем больше значение — тем медленнее анимация.

6. Загрузка экрана и цикл обновления LVGL

```
lv.scr_load(scr)

tim = time.ticks_ms()
while True:
    if time.ticks_ms() - tim > 5:
        tim = time.ticks_ms()
        lv.task_handler()
        lv.tick_inc(5)
```

LVGL требует периодического обновления каждые 5 мс для корректной отрисовки анимации.

4.3 Результаты эксперимента

Подключите модуль K210 к компьютеру с помощью microUSB-кабеля.

В среде CanMV IDE нажмите Connect, затем Run для запуска программы.

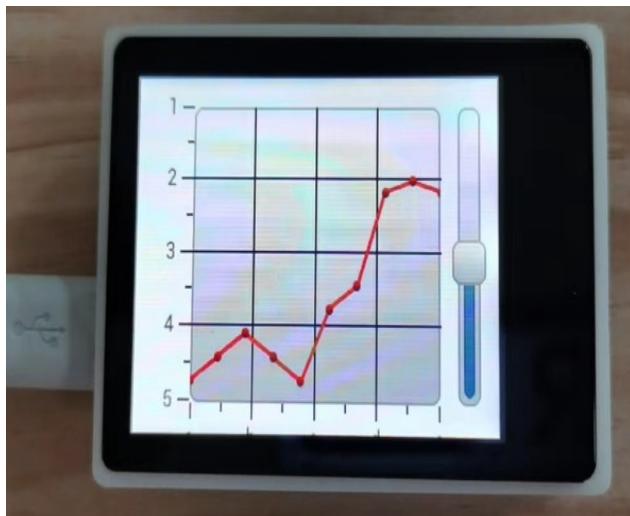
Также можно сохранить код как main.py и запускать его автономно.

В результате:

на экране отображается график;

кривая плавно изменяет диапазон;

скорость анимации регулируется слайдером справа.



4.4 Выводы по эксперименту

Анимация графика является достаточно сложной задачей, поэтому:

используются два отдельных класса, отвечающих за анимацию и сам график;
управление скоростью вынесено в отдельный элемент интерфейса — слайдер;
изменение скорости вступает в силу только после завершения текущего цикла анимации,
когда кривая достигает минимального значения.

Такой подход позволяет добиться плавной и стабильной анимации без
визуальных артефактов.

```

# -----
# Пример: Анимация графика (LVGL Chart Animation) + слайдер скорости
# Платформа: K210 / CanMV / MicroPython + LVGL
# -----


import lvgl as lv      # Основная библиотека LVGL (виджеты, стили, анимация)
import lvgl_helper as lv_h    # Вспомогательный модуль: "мост" LVGL ↔ дисплей/тач (flush/read)
import lcd            # Драйвер LCD-дисплея (инициализация, вывод)
import time           # Таймер/задержки/счётчики времени (ticks_ms)
import touchscreen as ts    # Драйвер тачскрина (инициализация)

# -----
# 1) Инициализация железа и LVGL
# -----


lcd.init()      # Включаем и настраиваем дисплей (без этого LVGL не сможет рисовать на LCD)
ts.init()        # Включаем и настраиваем тачскрин (чтобы LVGL получал касания/перетаскивания)
lv.init()        # Инициализация ядра LVGL (обязательно ДО создания любых объектов LVGL)

# -----
# 2) Настройка буфера отрисовки и драйвера дисплея LVGL
# -----


# LVGL рисует не "сразу на экран", а сначала в буфер (кусок памяти),
# а затем вызывает flush_cb, чтобы отправить нарисованные пиксели на экран.

disp_buf1 = lv.disp_buf_t()          # Структура буфера дисплея LVGL
buf1_1 = bytearray(320 * 10)        # Реальный буфер памяти под "полоску" экрана:
                                    # 320 пикселей по ширине * 10 строк (экономия RAM)
                                    # LVGL будет перерисовывать экран частями.

lv.disp_buf_init(
    disp_buf1,
    buf1_1,
    None,
    len(buf1_1) // 4                # Кол-во "пикселей" в буфере (часто делят на 4 в этих примерах)
)

disp_drv = lv.disp_drv_t()          # Драйвер дисплея (настройка параметров экрана)
lv.disp_drv_init(disp_drv)         # Заполняем драйвер значениями по умолчанию

disp_drv.buffer = disp_buf1        # Подключаем буфер к драйверу дисплея
disp_drv.flush_cb = lv_h.flush    # flush_cb — функция, которая реально выводит пиксели на LCD
disp_drv.hor_res = 320             # Горизонтальное разрешение дисплея
disp_drv.ver_res = 240             # Вертикальное разрешение дисплея

lv.disp_drv_register(disp_drv)     # Регистрируем дисплей в LVGL (после этого LVGL "видит" экран)

# -----
# 3) Настройка устройства ввода (тач) для LVGL
# -----


# LVGL работает с вводом через "in dev driver" (мышь/тач/энкодер и т.д.)

indev_drv = lv.indev_drv_t()       # Драйвер устройства ввода
lv.indev_drv_init(indev_drv)       # Значения по умолчанию

indev_drv.type = lv.INDEV_TYPE.POINTER  # POINTER = тач/мышь (координаты + нажатие)
indev_drv.read_cb = lv_h.read        # read_cb — функция, которая отдаёт LVGL координаты касания

lv.indev_drv_register(indev_drv)     # Регистрируем ввод

```

```

# -----
# 4) Класс Anim — удобная "обёртка" над lv.anim_t
# -----
# Зачем обёртка?
# Чтобы не писать каждый раз много однотипных вызовов lv.anim_set....
# Мы задаём:
# - диапазон значений (val → val+size),
# - время анимации,
# - как применять значение (exec_cb),
# - форму кривой (path_cb),
# - что делать по завершении (ready_cb),
# и сразу создаём анимацию.

class Anim(lv.anim_t):
    def __init__(self, obj, val, size, exec_cb, path_cb, time=500, playback=False, ready_cb=None):
        super().__init__() # Конструктор базового класса (lv.anim_t)

        lv.anim_init(self) # Инициализация структуры анимации
        lv.anim_set_time(self, time, 0) # Длительность анимации (ms). Второй параметр — задержка (delay)

        # Значения анимации: начальное val → конечное val + size
        lv.anim_set_values(self, val, val + size)

        # exec_cb — функция, которая будет вызвана на каждом "шаге" анимации.
        # Вариант 1: custom_exec_cb (если передали Python-функцию/lambda)
        # Вариант 2: стандартный exec_cb LVGL (если передали встроенный callback)
        if callable(exec_cb):
            lv.anim_set_custom_exec_cb(self, exec_cb)
        else:
            lv.anim_set_exec_cb(self, obj, exec_cb)

        lv.anim_set_path_cb(self, path_cb) # Кривая анимации (ease_in, ease_out, linear...)

        # playback = проигрывание назад (туда-сюда). В примере не используется, но оставлено как опция.
        if playback:
            lv.anim_set_playback(self, 0)

        # ready_cb — вызывается, когда анимация полностью завершилась
        if ready_cb:
            lv.anim_set_ready_cb(self, ready_cb)

        lv.anim_create(self) # Запуск/создание анимации в LVGL

# -----
# 5) Класс AnimatedChart — график с "вечной" анимацией диапазона
# -----
# Идея:
# - Мы не двигаем точки графика, а меняем "range" (диапазон шкалы Y).
# - В фазе 1 range увеличивается (ease_in).
# - В фазе 2 range уменьшается (ease_out).
# - Фазы вызывают друг друга через ready_cb, получаем бесконечный цикл.

class AnimatedChart(lv.chart):
    def __init__(self, parent, val, size):
        super().__init__(parent) # Создаём объект графика и прикрепляем к родителю (parent)

        # Параметры анимации:
        self.val = val # Базовое значение диапазона
        self.size = size # Насколько увеличивать/уменьшать диапазон
        self.max = 2000 # "Длинное" время (мс) для одной фазы (через factor)
        self.min = 500 # "Короткое" время (мс) для второй фазы (через factor)
        self.factor = 100 # Коэффициент скорости (меняется слайдером)

        self.anim_phase1() # Запускаем первую фазу сразу

    def anim_phase1(self):
        # Фаза 1: увеличиваем диапазон (например от 0..100 к 0..1100)
        Anim(
            self,
            self.val,
            self.size,
            # На каждом шаге: set_range(ось, значение)
            # В LVGL chart.set_range(axis, min, max), но в этой сборке используется self.set_range(0, val)
            lambda a, val: self.set_range(0, val),
            lv.anim_path_ease_in, # Плавный старт (ускорение)
            ready_cb=lambda a: self.anim_phase2(), # Когда дошли до верхней точки → запускаем фазу 2
            time=(self.max * self.factor) // 100 # Время зависит от factor (слайдер)
        )

    def anim_phase2(self):
        # Фаза 2: уменьшаем диапазон обратно
        Anim(
            self,
            self.val + self.size,
            -self.size, # отрицательный size — значит идём вниз
            lambda a, val: self.set_range(0, val),
            lv.anim_path_ease_out, # Плавное торможение (замедление)
            ready_cb=lambda a: self.anim_phase1(), # Дошли до низа → снова фазу 1
            time=(self.min * self.factor) // 100
        )

```



```

# -----
# 8) Слайдер для управления скоростью анимации
# -----
# Идея: factor влияет на время анимации в anim_phase1/phase2.
# Чем больше factor → тем дольше время → тем медленнее визуально анимация.

def on_slider_changed(self, obj=None, event=-1):
    # Здесь мы просто обновляем factor.
    # Важно понимать: текущая анимация уже создана и идёт по своему времени.
    # Поэтому изменение factor полностью проявится после завершения текущего цикла
    # (когда ready_cb запустит следующую фазу с новым временем).
    chart.factor = slider.get_value()

slider = lv.slider(scr)
slider.align(chart, lv.ALIGN.OUT_RIGHT_TOP, 10, 0) # Справа от графика
slider.set_width(30)
slider.set_height(chart.get_height())

slider.set_range(10, 200)                      # Минимум/максимум скорости
slider.set_value(chart.factor, 0)                # Стартовое значение = 100
slider.set_event_cb(on_slider_changed)          # Обработчик изменения

# -----
# 9) Загрузка экрана и главный цикл обновления LVGL
# -----
lv.scr_load(scr)                                # Делаем scr активным экраном

# LVGL требует регулярного вызова:
# - lv.task_handler() для обработки внутренних задач, перерисовок, анимации
# - lv.tick_inc(ms) чтобы LVGL знал, сколько "времени прошло"
# Здесь обновляем каждые ~5 мс, чтобы анимация была плавной.

tim = time.ticks_ms()
while True:
    if time.ticks_ms() - tim > 5:
        tim = time.ticks_ms()
        lv.task_handler()
        lv.tick_inc(5)

```

Мини-блок «Проверь себя» (по коду анимации графика)

Зачем в начале нужны lcd.init(), ts.init() и lv.init()?

Подумай: что будет, если убрать lv.init() или не инициализировать дисплей/тач?

Для чего создаётся буфер buf1_1 = bytearray(320*10)?

Что означает “320×10” и почему это выгоднее, чем буфер на весь экран 320×240?

Какую роль выполняет disp_drv.flush_cb = lv_h.flush?

Что именно делает flush_cb в момент отрисовки интерфейса?

Зачем нужен драйвер ввода indev_drv и почему type = lv.INDEV_TYPE.POINTER?

Что произойдёт со слайдером, если не зарегистрировать read_cb?

В классе Anim: что задают lv.anim_set_values(self, val, val+size) и exec_cb?

Сформулируй: “какие числа меняются” и “куда они применяются”.

Что именно анимируется в этом примере?

точки графика?

цвет линии?

диапазон оси/шкалы?

Укажи точный вызов, который изменяется во времени.

Почему у AnimatedChart две фазы: anim_phase1() и anim_phase2()?

Объясни, как получается бесконечный цикл анимации без while внутри класса.

Как factor влияет на скорость анимации?

Посмотри на формулы времени:

time = (self.max * self.factor) // 100 и time = (self.min * self.factor) // 100

При увеличении factor анимация ускоряется или замедляется — и почему?

Почему изменение скорости через слайдер может “вступать в силу” не сразу?

Какая часть кода объясняет, что новое время применяется только при создании следующей анимации?

Зачем в главном цикле нужны сразу две строки: lv.task_handler() и lv.tick_inc(5)?

Что будет, если оставить только одну из них?

5 Многовкладочная (многостраницчная) компоновка

5.1 Цель эксперимента

В данном эксперименте вы познакомитесь с созданием многовкладочной компоновки интерфейса (Multi-label layout).

Интерфейс состоит из трёх отдельных страниц (вкладок), между которыми можно переключаться.

Путь к эталонному коду данного эксперимента:

CanM -> 04-GUI -> multi_layout.py

5.2 Ход эксперимента

Заводская прошивка модуля уже содержит встроенную графическую библиотеку LVGL. Если ранее была установлена другая прошивка, перед выполнением эксперимента необходимо заново прошить модуль заводской прошивкой.

Сначала импортируются необходимые библиотеки, инициализируется LVGL, регистрируется драйвер дисплея и драйвер ввода (сенсорный экран).

Класс страницы с кнопкой — Page_Buttons

Создаётся класс страницы Page.Buttons, основная функция которого — отображение кнопки-счётчика.

При каждом нажатии на кнопку значение счётчика увеличивается на единицу и отображается на экране.

```
class Page.Buttons:  
    def __init__(self, app, page):  
        self.app = app  
        self.page = page  
  
        # Кнопка-счётчик  
        self.counter_btn = lv.btn(page)  
        self.counter_btn.set_size(100,60)  
        self.counter_btn.align(page, lv.ALIGN.CENTER, 0, 0)  
  
        self.counter_label = lv.label(self.counter_btn)  
        self.counter_label.set_text('Count')  
  
        self.counter_btn.set_event_cb(self.on_counter_btn)  
        self.counter = 0  
  
    def on_counter_btn(self, obj, event):  
        if event == lv.EVENT.CLICKED:  
            self.counter += 1  
            self.counter_label.set_text(str(self.counter))
```

Класс страницы со слайдером — Page_Slider

Создаётся класс Page_Slider, который отображает ползунок (slider) и в реальном времени показывает его текущее значение.

```
class Page_Slider:  
    def __init__(self, app, page):  
        self.app = app  
        self.page = page  
  
        # Слайдер  
        self.slider = lv.slider(page)  
        self.slider.align(page, lv.ALIGN.CENTER, 0, -10)  
  
        self.slider_label = lv.label(page)  
        self.slider_label.align(self.slider, lv.ALIGN.OUT_LEFT_MID, -10, 0)  
  
        self.slider.set_event_cb(self.on_slider_changed)  
        self.on_slider_changed(None)  
  
    def on_slider_changed(self, obj=None, event=-1):  
        self.slider_label.set_text(str(self.slider.get_value()))
```

Класс анимации — Anim

Создаётся универсальный класс Anim, который используется для создания анимаций LVGL с заданной траекторией, временем выполнения и обратным воспроизведением.

```
class Anim(lv.anim_t):  
    def __init__(self, obj, val, size, exec_cb, path_cb,  
                 time=500, playback=False, ready_cb=None):  
        super().__init__()  
        lv.anim_init(self)  
        lv.anim_set_time(self, time, 0)  
        lv.anim_set_values(self, val, val + size)  
  
        if callable(exec_cb):  
            lv.anim_set_custom_exec_cb(self, exec_cb)  
        else:  
            lv.anim_set_exec_cb(self, obj, exec_cb)  
  
        lv.anim_set_path_cb(self, path_cb)  
  
        if playback:  
            lv.anim_set_playback(self, 0)  
        if ready_cb:  
            lv.anim_set_ready_cb(self, ready_cb)  
  
    lv.anim_create(self)
```

Класс анимированного графика — AnimatedChart

Класс AnimatedChart наследуется от lv.chart и реализует циклическую анимацию изменения диапазона графика.

```
class AnimatedChart(lv.chart):
    def __init__(self, parent, val, size):
        super().__init__(parent)
        self.val = val
        self.size = size
        self.max = 2000
        self.min = 500
        self.factor = 100
        self.anim_phase1()

    def anim_phase1(self):
        Anim(
            self,
            self.val,
            self.size,
            lambda a, val: self.set_range(0, val),
            lv.anim_path_ease_in,
            ready_cb=lambda a: self.anim_phase2(),
            time=(self.max * self.factor) // 100
        )

    def anim_phase2(self):
        Anim(
            self,
            self.val + self.size,
            -self.size,
            lambda a, val: self.set_range(0, val),
            lv.anim_path_ease_out,
            ready_cb=lambda a: self.anim_phase1(),
            time=(self.min * self.factor) // 100
        )
```

Страница с графиком — Page_Chart

Класс Page_Chart предназначен для отображения анимированного графика.
Также добавлен вертикальный слайдер, который управляет скоростью анимации графика.

```
class Page_Chart:  
    def __init__(self, app, page):  
        self.app = app  
        self.page = page  
  
        self.chart = AnimatedChart(page, 100, 1000)  
        self.chart.set_width(page.get_width() - 100)  
        self.chart.set_height(page.get_height() - 30)  
        self.chart.align(page, lv.ALIGN.CENTER, 0, 0)  
  
        self.series1 = self.chart.add_series(lv.color_hex(0xFF0000))  
        self.chart.set_type(self.chart.TYPE.POINT | self.chart.TYPE.LINE)  
        self.chart.set_series_width(3)  
        self.chart.set_range(0, 100)  
        self.chart.init_points(self.series1, 10)  
        self.chart.set_points(  
            self.series1,  
            [10, 20, 30, 20, 10, 40, 50, 80, 95, 80]  
        )  
  
        self.chart.set_x_tick_texts('a\nb\nc\nd\ne', 2, lv.chart.AXIS.DRAW_LAST_TICK)  
        self.chart.set_x_tick_length(10, 5)  
        self.chart.set_y_tick_texts('1\n2\n3\n4\n5', 2, lv.chart.AXIS.DRAW_LAST_TICK)  
        self.chart.set_y_tick_length(10, 5)  
        self.chart.set_div_line_count(3, 3)  
        self.chart.set_margin(30)  
  
        self.slider = lv.slider(page)  
        self.slider.align(self.chart, lv.ALIGN.OUT_RIGHT_TOP, 10, 0)  
        self.slider.set_width(30)  
        self.slider.set_height(self.chart.get_height())  
        self.slider.set_range(10, 200)  
        self.slider.set_value(self.chart.factor, 0)  
        self.slider.set_event_cb(self.on_slider_changed)  
  
    # Слайдер управления скоростью анимации графика  
    def on_slider_changed(self, obj=None, event=-1):  
        self.chart.factor = self.slider.get_value()
```

Главный экран — Screen_Main

Класс Screen_Main использует элемент tabview для управления несколькими страницами интерфейса.

```
class Screen_Main(lv.obj):
    def __init__(self, app, *args, **kwds):
        self.app = app
        super().__init__(*args, **kwds)

        self.tabview = lv.tabview(self)
        self.tabview.set_style(lv.tabview.STYLE.BG, lv.style_plain_color)

        self.page_buttons = Page.Buttons(self.app, self.tabview.add_tab('Button'))
        self.page_slider = Page.Slider(self.app, self.tabview.add_tab('Slider'))
        self.page_chart = Page.Chart(self.app, self.tabview.add_tab('Chart'))
```

Загрузка главного экрана

```
screen_main = Screen_Main(lv.obj())
lv.scr_load(screen_main)
```

Обновление задач LVGL

Поскольку элементы LVGL требуют регулярного обновления, обработчик задач LVGL вызывается каждые 5 мс.



5.3 Результаты эксперимента

Подключите модуль K210 к компьютеру с помощью кабеля microUSB.

В среде CanMV IDE нажмите кнопку подключения, затем кнопку Run для запуска программы. Также возможно сохранить код как main.py и запустить его непосредственно на модуле K210.

В результате на верхней части LCD-экрана отображаются три вкладки:

Button — страница с кнопкой-счётчиком

Slider — страница со слайдером

Chart — страница с динамическим графиком

При касании соответствующей вкладки происходит переход на нужную страницу.

Функциональность каждой страницы соответствует ранее рассмотренным примерам, но теперь они объединены в единый многовкладочный интерфейс.

Алгоритм

Практичный алгоритм разработки многостраничных приложений на LVGL (MicroPython) для K210/CanMV. Это именно последовательность шагов, по которой удобно собирать проекты с вкладками/страницами.

1) Подготовить каркас проекта

Определить структуру файлов:

main.py — запуск, инициализация, главный экран, цикл LVGL
pages/ (по желанию) — отдельные файлы страниц: page_buttons.py, page_slider.py, page_chart.py
widgets/ (по желанию) — переиспользуемые компоненты: индикаторы, графики, меню

Сразу выбрать модель навигации:

TabView (вкладки сверху/снизу) — проще всего

Менеджер экранов (несколько lv.obj() и lv.scr_load()) — для полноэкранных “сцен”

Контейнер + переключение видимости — для быстрых переходов без scr_load

2) Инициализация железа и LVGL (один раз)

lcd.init(), ts.init(), lv.init()

Настроить:

буфер дисплея lv.disp_buf_t()

драйвер дисплея lv.disp_drv_t() + flush_cb

драйвер ввода lv.indev_drv_t() + read_cb

Вынести инициализацию в функцию init_lvgl() чтобы главный код был чистым.

3) Ввести “контекст приложения” (App Context)

Сделать объект/словарь app, который хранит общие вещи:

ссылки на дисплей/ввод (если надо)

общие стили (цвета, шрифты)

общие данные (настройки, состояния)

методы навигации: go(page_name), back()

Идея: страницы не должны “знать” про железо — только про app и свой parent.

4) Определить единый интерфейс страницы

Для каждой страницы завести одинаковый набор методов (не обязательно все, но удобно):

build(parent) — создать виджеты

on_show() — когда страница стала активной (обновить данные)

on_hide() — когда уходим со страницы (остановить анимации/таймеры)

destroy() — если нужно реально удалить объекты и освободить память

5) Создать навигационный контейнер

Выбрать один вариант:

Вариант А: TabView

```
tabview = lv.tabview(root)
tabview.add_tab("Button"), add_tab("Slider"), add_tab("Chart")
На каждой вкладке строится своя страница.
```

Вариант В: Screen manager

страницы строятся в отдельные lv.obj()/lv.scr_act() контейнеры
переключение через lv.scr_load(new_screen) или скрытие/показ контейнеров.
Для обучающих проектов на K210 чаще всего лучший старт — TabView.

6) Реализовать страницы как классы

В конструкторе принимать: app, parent

Внутри создать все элементы, подписаться на события:

кнопки — set_event_cb
слайдеры — set_event_cb

События должны менять только состояние страницы и/или app.

7) События и связь между страницами

Обновление элементов из события — ок.

Передача данных между страницами — через app:

```
app.state["speed"] = value
другая страница читает это в on_show()
```

8) Анимации/таймеры: отдельное управление жизненным циклом

На K210 важно не плодить бесконечные анимации без контроля.

Если на странице есть анимация:

запускать в on_show()
останавливать в on_hide() (если библиотека/версия позволяет)
или делать флаг self.active = True/False и в callback проверять активность.

9) Главный цикл LVGL (обязателен для MicroPython)

Каждые ~5 мс:

```
lv.task_handler()
lv.tick_inc(5)
```

Это сердце интерфейса. Без него не будет плавной перерисовки и корректной работы событий.

10) Оптимизация под K210 (чтобы не “падало”)

Не создавать/удалять крупные объекты постоянно — лучше:

создать один раз и переиспользовать
скрывать/показывать контейнеры

Стараться:

меньше теней/сложных стилей
не перегружать графики большим числом точек
Держать буфер дисплея разумным (как в примерах 320*10 строк).

11) Проверка и отладка

Проверить каждую страницу отдельно:

запускается ли
есть ли события
нет ли утечки (после 50–100 переключений)

Логировать ключевые события в UART:

“tab changed”, “slider value”, “start anim”, “stop anim”.

12) Финализация: упаковка в “приложение”

```
screen_main = Screen_Main(app)
```

```
lv.scr_load(screen_main)
```

запуск цикла LVGL

сохранить как main.py и проверить запуск “без IDE”.

Полный шаблон main.py

```
import lvgl as lv
import lvgl_helper as lv_h
import lcd
import time
import touchscreen as ts

# =====
# Инициализация железа
# =====

lcd.init()
ts.init()
lv.init()

# Буфер дисплея
disp_buf = lv.disp_buf_t()
buf1 = bytearray(320 * 10)
lv.disp_buf_init(disp_buf, buf1, None, len(buf1) // 4)

# Драйвер дисплея
disp_drv = lv.disp_drv_t()
lv.disp_drv_init(disp_drv)
disp_drv.buffer = disp_buf
disp_drv.flush_cb = lv_h.flush
disp_drv.hor_res = 320
disp_drv.ver_res = 240
lv.disp_drv_register(disp_drv)

# Драйвер ввода (тач)
indev_drv = lv.indev_drv_t()
lv.indev_drv_init(indev_drv)
indev_drv.type = lv.INDEV_TYPE.POINTER
indev_drv.read_cb = lv_h.read
lv.indev_drv_register(indev_drv)

# =====
# Контекст приложения
# =====

class App:
    def __init__(self):
        self.state = {} # общее состояние между страницами

app = App()

# =====
# Базовый класс страницы
# =====

class BasePage:
    def __init__(self, app, parent):
        self.app = app
        self.parent = parent
        self.build()

    def build(self):
        pass

    def on_show(self):
        pass

    def on_hide(self):
        pass
```

```
# =====
# Страница 1: Кнопка
# =====

class PageButtons(BasePage):
    def build(self):
        self.counter = 0

        self.btn = lv.btn(self.parent)
        self.btn.set_size(120, 60)
        self.btn.align(self.parent, lv.ALIGN.CENTER, 0, 0)
        self.btn.set_event_cb(self.on_click)

        self.label = lv.label(self.btn)
        self.label.set_text("0")

    def on_click(self, obj, event):
        if event == lv.EVENT.CLICKED:
            self.counter += 1
            self.label.set_text(str(self.counter))

# =====
# Страница 2: Слайдер
# =====

class PageSlider(BasePage):
    def build(self):
        self.slider = lv.slider(self.parent)
        self.slider.set_width(200)
        self.slider.align(self.parent, lv.ALIGN.CENTER, 0, 0)
        self.slider.set_event_cb(self.on_change)

        self.label = lv.label(self.parent)
        self.label.align(self.slider, lv.ALIGN.OUT_BOTTOM_MID, 0, 10)
        self.on_change(None)

    def on_change(self, obj=None, event=-1):
        value = self.slider.get_value()
        self.label.set_text("Value: {}".format(value))
        self.app.state["slider"] = value
```

```
# =====
# Страница 3: График (заглушка)
# =====

class PageChart(BasePage):
    def build(self):
        self.label = lv.label(self.parent)
        self.label.set_text("Chart page")
        self.label.align(self.parent, lv.ALIGN.CENTER, 0, 0)

    def on_show(self):
        print("Chart page active")

# =====
# Главный экран с вкладками
# =====

class ScreenMain(lv.obj):
    def __init__(self, app):
        super().__init__()
        self.app = app

        self.tabview = lv.tabview(self)

        self.tab_btn = self.tabview.add_tab("Button")
        self.tab_sld = self.tabview.add_tab("Slider")
        self.tab_cht = self.tabview.add_tab("Chart")

        self.page_buttons = PageButtons(app, self.tab_btn)
        self.page_slider = PageSlider(app, self.tab_sld)
        self.page_chart = PageChart(app, self.tab_cht)

# =====
# Запуск
# =====

screen = ScreenMain(app)
lv.scr_load(screen)

# =====
# Главный цикл LVGL
# =====

tick = time.ticks_ms()

while True:
    if time.ticks_ms() - tick >= 5:
        tick = time.ticks_ms()
        lv.tick_inc(5)
        lv.task_handler()
```

Контрольная работа: процедуры графических операций (LVGL + K210)

Секундомер HH:MM:SS + кнопка START/STOP

```
import lvgl as lv
import lvgl_helper as lv_h
import lcd
import touchscreen as ts
import time

# -----
# Инициализация
# -----
lcd.init()
ts.init()
lv.init()

disp_buf = lv.disp_buf_t()
buf = bytearray(320 * 10)
lv.disp_buf_init(disp_buf, buf, None, len(buf) // 4)

disp_drv = lv.disp_drv_t()
lv.disp_drv_init(disp_drv)
disp_drv.buffer = disp_buf
disp_drv.flush_cb = lv_h.flush
disp_drv.hor_res = 320
disp_drv.ver_res = 240
lv.disp_drv_register(disp_drv)

indev_drv = lv.indev_drv_t()
lv.indev_drv_init(indev_drv)
indev_drv.type = lv.INDEV_TYPE.POINTER
indev_drv.read_cb = lv_h.read
lv.indev_drv_register(indev_drv)

scr = lv.scr_act()

# -----
# Секундомер
# -----
hours = 0
minutes = 0
seconds = 0
running = False

# Метка времени
time_label = lv.label(scr)
time_label.set_text("00:00:00")

# LVGL v7-совместимое выравнивание
try:
    time_label.align(None, lv.ALIGN.IN_TOP_MID, 0, 40)
except:
    # если align() другой — просто оставим по умолчанию
    pass

def set_time_text():
    time_label.set_text("{:02d}:{:02d}:{:02d}".format(hours, minutes, seconds))

# Кнопка START/STOP
def btn_event_handler(obj, event):
    global running
    if event == lv.EVENT.CLICKED:
        running = not running
        if running:
            btn_label.set_text("STOP")
        else:
            btn_label.set_text("START")

btn = lv.btn(scr)
btn.set_size(120, 60)

try:
    btn.align(None, lv.ALIGN.IN_BOTTOM_MID, 0, -30)
except:
    pass

btn.set_event_cb(btn_event_handler)

btn_label = lv.label(btn)
btn_label.set_text("START")
btn_label.align(None, lv.ALIGN.CENTER, 0, 0)

# -----
# Главный цикл + обновление раз в 1 секунду
# -----
lv_tick_ms = time.ticks_ms()
sw_tick_ms = time.ticks_ms()

while True:
    # LVGL системный тик (как в твоём примере)
    if time.ticks_ms() - lv_tick_ms > 5:
        lv_tick_ms = time.ticks_ms()
        lv.task_handler()
        lv.tick_inc(5)

    # Секундомер: обновляем ровно раз в 1000 мс
    if running and (time.ticks_ms() - sw_tick_ms >= 1000):
        sw_tick_ms = time.ticks_ms()

        seconds += 1
        if seconds >= 60:
            seconds = 0
            minutes += 1
        if minutes >= 60:
            minutes = 0
            hours += 1
        if hours >= 24:
            hours = 0

    set_time_text()
```

5. Визуальные эксперименты с ИИ

1. Распознавание цвета

1.1 Цели эксперимента

В данном эксперименте рассматривается функция распознавания цвета.

Система выделяет объекты одного цвета на изображении на основе значений цвета в пространстве LAB.

Путь к эталонному коду эксперимента:

CanMV -> 05-AI -> color_recognition.py

1.2 Ход эксперимента

Заводская прошивка модуля уже содержит интегрированный модуль алгоритмов AI-зрения.

Если ранее была установлена другая прошивка, перед выполнением эксперимента необходимо заново прошить модуль заводской прошивкой.

Импортируем необходимые библиотеки и инициализируем камеру и LCD-дисплей:

```
import sensor, image, time, lcd
```

```
lcd.init()  
sensor.reset()  
sensor.set_pixformat(sensor.RGB565)  
sensor.set_framesize(sensor.QVGA)  
sensor.skip_frames(time = 2000)  
sensor.set_auto_gain(False)  
sensor.set_auto_whitebal(False)  
clock = time.clock()
```

На экране камеры рисуется белая рамка размером 50×50 пикселей.

Она служит подсказкой пользователю — в неё необходимо поместить цвет, который будет распознаваться.

```
r = [(320//2)-(50//2), (240//2)-(50//2), 50, 50]  
for i in range(50):
```

```
    img = sensor.snapshot()  
    img.draw_rectangle(r)  
    lcd.display(img)
```

Когда белая рамка сменяется на зелёную, система начинает обучение — вычисляется LAB-значение цвета внутри рамки.

Значения считаются несколько раз, после чего усредняются для получения итогового порога цвета.

```

print("Learning thresholds...")
threshold = [50, 50, 0, 0, 0, 0] # Средние значения L, A, B
for i in range(50):
    img = sensor.snapshot()
    hist = img.get_histogram(roi=r)
    lo = hist.get_percentile(0.01) # 1% перцентиль (при необходимости можно изменить)
    hi = hist.get_percentile(0.99) # 99% перцентиль
    # Усреднение значений
    threshold[0] = (threshold[0] + lo.l_value()) // 2
    threshold[1] = (threshold[1] + hi.l_value()) // 2
    threshold[2] = (threshold[2] + lo.a_value()) // 2
    threshold[3] = (threshold[3] + hi.a_value()) // 2
    threshold[4] = (threshold[4] + lo.b_value()) // 2
    threshold[5] = (threshold[5] + hi.b_value()) // 2
    for blob in img.find_blobs([threshold], pixels_threshold=100, area_threshold=100, merge=True, margin=10):
        img.draw_rectangle(blob.rect())
        img.draw_cross(blob.cx(), blob.cy())
        img.draw_rectangle(r, color=(0,255,0))
    lcd.display(img)

```

После завершения обучения запускается бесконечный цикл, в котором система анализирует изображение с камеры и сравнивает текущие LAB-значения с ранее обученными.

Если значения совпадают с порогом, соответствующая область выделяется рамкой.

```

print("Thresholds learned...")
print("Start Color Recognition...")
while(True):
    clock.tick()
    img = sensor.snapshot()
    for blob in img.find_blobs([threshold], pixels_threshold=100, area_threshold=100, merge=True, margin=10):
        img.draw_rectangle(blob.rect())
        img.draw_cross(blob.cx(), blob.cy())
    lcd.display(img)
    print(clock.fps())

```

1.3 Результаты эксперимента

Подключите модуль K210 к компьютеру с помощью кабеля microUSB.

В CanMV IDE нажмите кнопку Connect, затем Run, чтобы запустить программу.

Также возможно сохранить код как main.py и запустить его напрямую на модуле K210.

После завершения инициализации система выводит изображение с камеры на LCD-экран.

В центре экрана отображается белая рамка — в течение примерно 3 секунд в неё необходимо поместить цвет для распознавания.

Когда белая рамка становится зелёной, начинается процесс обучения цвета по LAB-модели.

На экране могут появляться дополнительные рамки в качестве визуального эффекта.

Примерно через 5 секунд зелёная рамка исчезает — это означает, что обучение завершено.

После этого направьте камеру на объект нужного цвета — система автоматически выделит распознанный цвет рамкой.



1.4 Итоги эксперимента

Функция распознавания цвета основана на анализе значений цвета в пространстве LAB. Сначала цвет помещается в заданную область, затем система обучается на основе LAB-значений, считанных внутри этой области.

В процессе работы полученные значения сравниваются с текущим изображением с камеры, и при совпадении цвет считается распознанным и выделяется рамкой.

Следует учитывать, что точность распознавания зависит от контраста между цветом объекта и фоном.

Если цвет объекта близок к цвету фона, вероятность ошибочного распознавания значительно возрастает.

2. Распознавание штрихкодов (Barcode recognition)

2.1 Цели эксперимента

В этом эксперименте изучается функция распознавания штрихкодов: модуль находит штрихкод на изображении, выделяет его рамкой и выводит в терминал данные (тип, содержимое, угол поворота, качество и FPS).

Путь к примеру кода: CanMV\05-AI\find_barcodes.py

2.2 Процедура эксперимента

В заводскую прошивку модуля уже интегрированы алгоритмы компьютерного зрения. Если вы устанавливали другую прошивку, перед экспериментом прошивайте модуль обратно заводской прошивкой.

Импортируйте библиотеки и инициализируйте камеру и LCD. В данном примере камера может работать в RGB565 (как в коде) или в оттенках серого (GRAYSCALE) — это отмечено в комментарии.

```
import sensor, image, time, math, lcd
```

```
lcd.init()
sensor.reset()
sensor.set_pixformat(sensor.RGB565) #GRAYSCALE
sensor.set_framesize(sensor.QVGA)
sensor.skip_frames(time = 100)
sensor.set_auto_gain(False)
sensor.set_auto_whitebal(False)
clock = time.clock()
```

Тип штрихкода в библиотеке возвращается как значение (enum), поэтому создаётся функция, которая преобразует тип штрихкода в строку (название типа):

```
def barcode_name(code):
    if(code.type() == image.EAN2):
        return "EAN2"
    if(code.type() == image.EAN5):
        return "EAN5"
    if(code.type() == image.EAN8):
        return "EAN8"
    if(code.type() == image.UPCE):
        return "UPCE"
    if(code.type() == image.ISBN10):
        return "ISBN10"
    if(code.type() == image.UPCA):
        return "UPCA"
    if(code.type() == image.EAN13):
        return "EAN13"
    if(code.type() == image.ISBN13):
        return "ISBN13"
    if(code.type() == image.I25):
        return "I25"
    if(code.type() == image.DATABAR):
        return "DATABAR"
    if(code.type() == image.DATABAR_EXP):
        return "DATABAR_EXP"
    if(code.type() == image.CODABAR):
        return "CODABAR"
    if(code.type() == image.CODE39):
        return "CODE39"
    if(code.type() == image.PDF417):
        return "PDF417"
    if(code.type() == image.CODE93):
        return "CODE93"
    if(code.type() == image.CODE128):
        return "CODE128"
```

Далее запускается бесконечный цикл: камера делает снимок, изображение анализируется функцией `find_barcodes()`. Если штрихкод найден — он обводится прямоугольником, а информация выводится в терминал. Также на экран выводится текущий FPS.

```
while(True):
    clock.tick()
    img = sensor.snapshot()
    fps = clock.fps()
    codes = img.find_barcodes()
    for code in codes:
        img.draw_rectangle(code.rect())
        print_args = (barcode_name(code), code.payload(), (180 * code.rotation()) / math.pi, code.quality(), fps)
        print("Barcode %s, Payload \"%s\", rotation %f (degrees), quality %d, FPS %f" % print_args)
        img.draw_string(0, 0, "%2.1ffps" %(fps), color=(0, 60, 128), scale=2.0)
    lcd.display(img)
```

Пояснение к выводимым данным:

Barcode %s — тип штрихкода (EAN13, CODE128 и т.д.).

Payload — содержимое (строка/данные, закодированные в штрихкоде).

rotation — угол поворота штрихкода (в градусах).

quality — оценка качества распознавания.

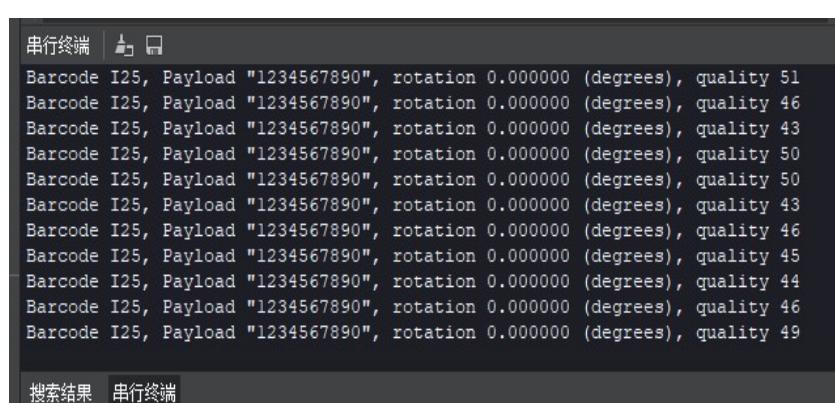
FPS — скорость обработки кадров.

2.3 Результаты эксперимента

Подключите модуль K210 к компьютеру через microUSB. В CanMV IDE нажмите Connect, затем Run для запуска примера. Также можно сохранить скрипт как main.py и запускать его автономно на модуле.

После завершения инициализации на LCD появится изображение с камеры. Когда камера «увидит» штрихкод:

штрихкод будет обведён рамкой на экране,
в Serial Terminal (внизу IDE) появится строка с параметрами распознанного штрихкода.



2.4 Итоги эксперимента

В этом эксперименте модуль распознаёт несколько форматов штрихкодов.

Для проверки можно:

найти в интернете генератор штрихкодов и создать штрихкод с произвольным текстом/числом;

либо в CanMV IDE открыть: Tools → Machine Vision → Barcode Generator (откроется страница/инструмент генерации).

3 Распознавание QR-кодов (QR Code Recognition)

3.1 Цель эксперимента

В данном эксперименте изучается функция распознавания двумерных кодов (QR-кодов). Алгоритм определяет наличие QR-кода в изображении, выделяет его рамкой и выводит информацию, содержащуюся в QR-коде, в последовательный терминал.

Эталонный путь к коду данного эксперимента:

CanMV\05-AI\find_qrcodes.py

3.2 Процедура эксперимента

Заводская прошивка модуля уже содержит встроенный модуль алгоритмов компьютерного зрения. Если ранее была загружена другая прошивка, перед выполнением эксперимента необходимо перепрошить модуль заводской прошивкой.

Импортируйте необходимые библиотеки и инициализируйте камеру и LCD-дисплей:

```
import sensor, image, time, lcd
```

```
lcd.init()
sensor.reset()
sensor.set_pixformat(sensor.RGB565)
sensor.set_framesize(sensor.QVGA)
sensor.skip_frames(time = 100)
```

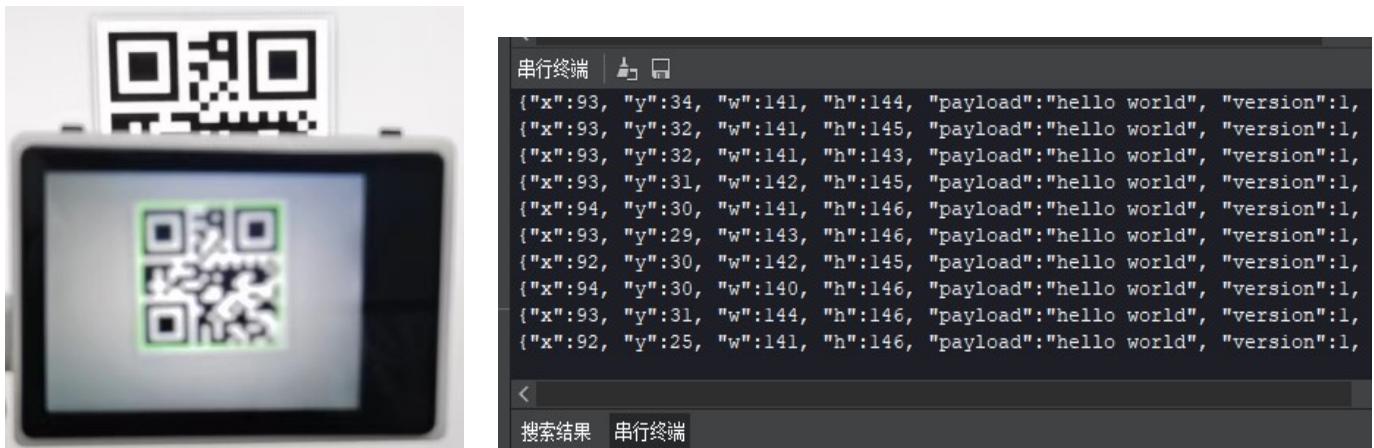
Используйте встроенную функцию `find_qrcodes()` для поиска QR-кодов на изображении. Если QR-код обнаружен, он выделяется прямоугольной рамкой, а информация из QR-кода выводится в терминал:

```
clock = time.clock()
while(True):
    clock.tick()
    img = sensor.snapshot()
    for code in img.find_qrcodes():
        img.draw_rectangle(code.rect(), color = 127, thickness = 3)
        print(code)
    lcd.display(img)
    # print(clock.fps())
```

3.3 Результаты эксперимента

Подключите модуль K210 к компьютеру с помощью кабеля microUSB. В среде CanMV IDE нажмите кнопку Connect, после установления соединения нажмите Run для запуска программы. Также можно сохранить код в файл main.py, загрузить его в модуль K210 и запускать автономно.

После завершения инициализации системы на LCD-дисплее отображается изображение с камеры. При попадании QR-кода в поле зрения камеры он выделяется рамкой, а в нижней части окна IDE, в последовательном терминале, выводится информация, закодированная в QR-коде.



3.4 Выводы по эксперименту

Эксперимент по распознаванию QR-кодов позволяет считывать информацию, содержащуюся в тестовых изображениях. Также можно использовать онлайн-генераторы QR-кодов для создания собственных QR-кодов с произвольной информацией и проверять их распознавание с помощью модуля K210.



4. Распознавание AprilTag

AprilTag — это визуальный маркер (похоже на QR-код), который компьютер или робот может легко найти и распознать с камеры. Его используют в робототехнике, компьютерном зрении и AR, когда нужно точно понять где находится объект и как он повернут.

Проще говоря:

AprilTag — это “якорь” в реальном мире для камеры.

Как выглядит Чёрно-белый квадрат

Внутри — уникальный узор (код)

Чёткая рамка → легко детектируется даже при плохом освещении

Что он умеет давать

Когда камера видит AprilTag, алгоритм сразу получает:

ID метки (какая именно)

Позицию (X, Y, Z) относительно камеры

Поворот (roll, pitch, yaw)

То есть не просто «вижу картинку», а «знаю, где она в пространстве».

Где используется

Очень популярная штука

Роботы

навигация (робот видит метки как ориентиры)

точная парковка и стыковка

Дроны

посадка на платформу

стабилизация

AR / VR

привязка виртуальных объектов к реальному миру

Обучение и соревнования

ROS

FTC / FRC

университетские проекты

Компьютерное зрение

калибровка камер

трекинг объектов

Алгоритм говорит:

“Метка №5 находится в 1.2 м впереди, смещена на 20 см вправо и повернута на 10°”

Всё — без GPS, только камера.

Популярные библиотеки

AprilTag 2 / 3 (официальная, C/C++)

OpenCV (через модуль contrib)

ROS apriltag

Python биндинги

4.1 Цель эксперимента

В данном эксперименте изучается функция распознавания машинных кодов (AprilTag). После обнаружения машинного кода он выделяется рамкой, а соответствующие данные выводятся в терминал.

Эталонный путь к коду эксперимента:
CanMV\05-AI\find_apriltags.py

4.2 Процедура эксперимента

Заводская прошивка модуля уже содержит алгоритмы компьютерного зрения. Если ранее была загружена другая прошивка, перед началом эксперимента необходимо заново прошить модуль заводской прошивкой.

Импортируйте необходимые библиотеки и инициализируйте камеру и LCD-дисплей:

```
import sensor, image, time, math, lcd
```

```
lcd.init()
sensor.reset()
sensor.set_pixformat(sensor.RGB565)
sensor.set_framesize(sensor.QQVGA)
sensor.skip_frames(time = 100)
sensor.set_auto_gain(False)
sensor.set_auto_whitebal(False)
clock = time.clock()
```

Задайте семейства машинных кодов, которые необходимо распознавать. Если какое-либо семейство не требуется, соответствующую строку можно закомментировать.

По умолчанию используется семейство TAG36H11.

```
tag_families = 0
tag_families |= image.TAG16H5 # закомментировать, чтобы отключить
tag_families |= image.TAG25H7 # закомментировать, чтобы отключить
tag_families |= image.TAG25H9 # закомментировать, чтобы отключить
tag_families |= image.TAG36H10 # закомментировать, чтобы отключить
tag_families |= image.TAG36H11 # закомментировать, чтобы отключить
# (семейство по умолчанию)
tag_families |= image.ARTOOLKIT # закомментировать, чтобы отключить
```

Создайте функцию для преобразования типа семейства машинного кода в строку:

```

def family_name(tag):
    if(tag.family() == image.TAG16H5):
        return "TAG16H5"
    if(tag.family() == image.TAG25H7):
        return "TAG25H7"
    if(tag.family() == image.TAG25H9):
        return "TAG25H9"
    if(tag.family() == image.TAG36H10):
        return "TAG36H10"
    if(tag.family() == image.TAG36H11):
        return "TAG36H11"
    if(tag.family() == image.ARTOOLKIT):
        return "ARTOOLKIT"

```

Создайте бесконечный цикл while, в котором вызывается функция find_apriltags() для поиска машинных кодов на изображении.

При обнаружении код выделяется рамкой, рисуется перекрестие в центре и выводится информация в терминал.

```

while(True):
    clock.tick()
    img = sensor.snapshot()
    #img = img.resize(280, 195)
    #img = img.resize(292, 210)
    for tag in img.find_apriltags(families=tag_families):
        img.draw_rectangle(tag.rect(), color = (255, 0, 0))
        img.draw_cross(tag.cx(), tag.cy(), color = (0, 255, 0))
        print_args = (family_name(tag), tag.id(), (180 * tag.rotation()) / math.pi)
        print("Tag Family %s, Tag ID %d, rotation %f (degrees)" % print_args)
    lcd.display(img)
    #print(clock.fps())

```

4.3 Результаты эксперимента

Подключите модуль K210 к компьютеру с помощью кабеля microUSB.

В CanMV IDE нажмите кнопку Connect, после успешного подключения нажмите Run для запуска программы. Также можно сохранить код как main.py и загрузить его в модуль K210 для автономного запуска.

После завершения инициализации на LCD-дисплее отображается изображение с камеры.

При наведении камеры на AprilTag выполняется распознавание следующих типов кодов:

TAG16H5
 TAG25H7
 TAG25H9
 TAG36H10
 TAG36H11
 ARTOOLKIT

Из-за ограниченной вычислительной мощности и объема памяти K210 обработка AprilTag требует значительных ресурсов, поэтому использование полного экранного разрешения в данный момент невозможно.

Обнаруженный машинный код выделяется рамкой, а информация о нем выводится в последовательный терминал в нижней части CanMV IDE.



```
Tag Family TAG36H11, Tag ID 16, rotation 1.271711 (degrees)
Tag Family TAG36H11, Tag ID 17, rotation 1.350201 (degrees)
Tag Family TAG36H11, Tag ID 16, rotation 1.330411 (degrees)
Tag Family TAG36H11, Tag ID 17, rotation 1.422991 (degrees)
Tag Family TAG36H11, Tag ID 16, rotation 1.085476 (degrees)
Tag Family TAG36H11, Tag ID 17, rotation 1.621595 (degrees)
Tag Family TAG36H11, Tag ID 16, rotation 1.380341 (degrees)
Tag Family TAG36H11, Tag ID 17, rotation 1.286299 (degrees)
Tag Family TAG25H9, Tag ID 4, rotation 271.907377 (degrees)
Tag Family TAG36H11, Tag ID 16, rotation 1.340031 (degrees)
Tag Family TAG36H11, Tag ID 17, rotation 1.733174 (degrees)
```

4.4 Итоги эксперимента

Распознавание машинных кодов требует большого объема вычислений, а ресурсы K210 ограничены, поэтому невозможно использовать полноэкранное разрешение.

Если требуется увеличить размер изображения, можно установить:

```
sensor.set_framesize(sensor.QVGA)
```

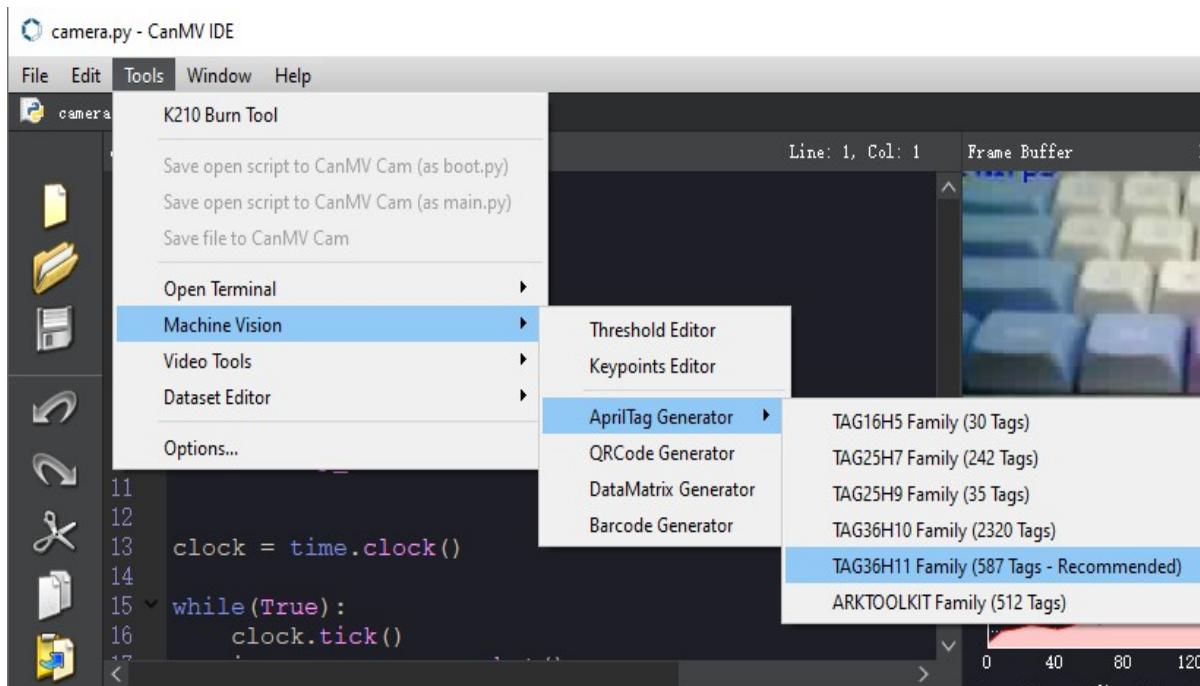
и после захвата изображения изменить его размер, например:

```
img = img.resize(292, 210)
```

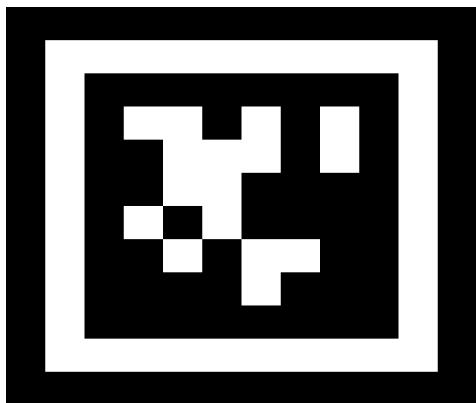
Однако это приведет к снижению частоты кадров и скорости распознавания, поэтому использовать данный метод следует с осторожностью.

По умолчанию в эксперименте используется семейство машинных кодов TAG36H11. Для генерации других типов AprilTag в CanMV IDE выберите:

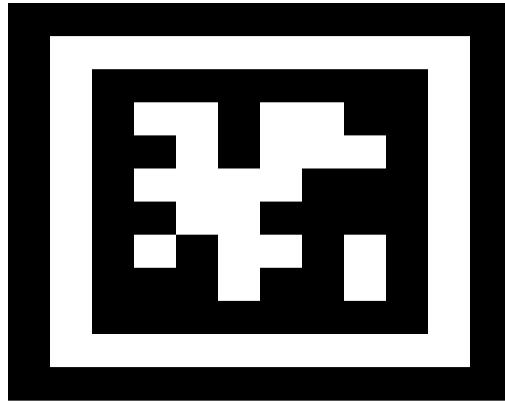
Tools → Machine Vision → AprilTag machine code generator



Примеры машинных кодов TAG36H11:



TAG36H11 – 0



TAG36H11 – 1

5 Распознавание лиц (Face Detection)

5.1 Цели эксперимента

В данном эксперименте изучается функция распознавания лиц. Камера захватывает изображение, после чего оно анализируется нейросетевой моделью. При обнаружении лица система выделяет его рамкой и выводит соответствующую информацию в терминал.

Путь к эталонному коду эксперимента:

CanMV\05-AI\yolo_face_detect.py

5.2 Подготовка к эксперименту

Перед началом эксперимента необходимо скопировать файл модели на карту памяти (SD), после чего вставить карту памяти в слот модуля K210.

5.3 Ход эксперимента

Заводская прошивка модуля уже содержит встроенный модуль алгоритмов компьютерного зрения. Если ранее была установлена другая прошивка, перед выполнением эксперимента необходимо перепрошить модуль заводской версией.

Инициализация камеры и дисплея

```
import sensor, image, time, lcd  
from maix import KPU
```

```
lcd.init()  
sensor.reset()  
sensor.set_pixformat(sensor.RGB565)  
sensor.set_framesize(sensor.QVGA)  
sensor.skip_frames(time = 100)  
clock = time.clock()
```

Инициализация KPU и загрузка модели

В бесконечном цикле изображение с камеры передаётся в KPU, обрабатывается нейросетью YOLOv2.

```
od_img = image.Image(size=(320,256))

anchor = (
    0.893, 1.463, 0.245, 0.389, 1.55, 2.58,
    0.375, 0.594, 3.099, 5.038, 0.057, 0.090,
    0.567, 0.904, 0.101, 0.160, 0.159, 0.255
)

kpu = KPU()
kpu.load_kmodel("/sd/KPU/yolo_face_detect/yolo_face_detect.kmodel")
kpu.init_yolo2(
    anchor,
    anchor_num=9,
    img_w=320,
    img_h=240,
    net_w=320,
    net_h=256,
    layer_w=10,
    layer_h=8,
    threshold=0.7,
    nms_value=0.3,
    classes=1
)
```

Основной цикл обработки

Для работы KPU необходимо загрузить файл нейросетевой модели (kmodel).

Путь к модели, используемой в данном эксперименте:

/sd/KPU/yolo_face_detect/yolo_face_detect.kmodel

Используется алгоритм YOLOv2 для сопоставления изображения с моделью.

Переменная od_img — это изображение размером 320×256, которое используется как вход нейросети.

```

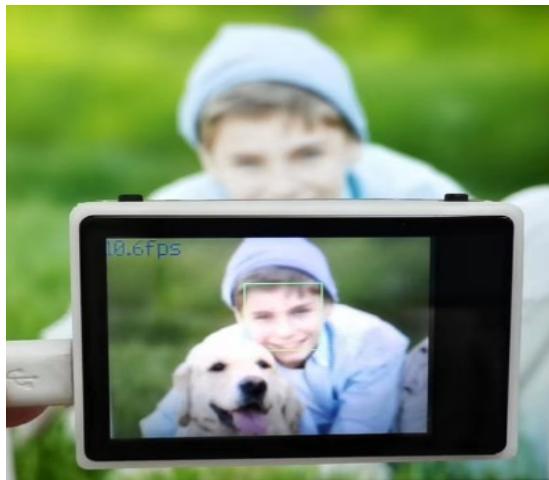
while True:
    clock.tick()
    img = sensor.snapshot()
    a = od_img.draw_image(img, 0, 0)
    od_img.pix_to_ai()
    kpu.run_with_output(od_img)
    dect = kpu.regionlayer_yolo2()
    fps = clock.fps()

    if len(dect) > 0:
        print("dect:", dect)
        for l in dect:
            img.draw_rectangle(l[0], l[1], l[2], l[3], color=(0, 255, 0))

    img.draw_string(0, 0, "%2.1ffps" % fps, color=(0, 60, 128), scale=2.0)
    lcd.display(img)

kpu.deinit()

```



```

dect: [[134, 54, 78, 82, 0, 0.7314994]]
dect: [[134, 53, 79, 83, 0, 0.7314994]]
dect: [[122, 39, 99, 109, 0, 0.7744358]]
dect: [[126, 39, 92, 110, 0, 0.7876179]]
dect: [[127, 39, 91, 110, 0, 0.8548551]]
dect: [[126, 38, 92, 110, 0, 0.7744358]]
dect: [[122, 38, 99, 110, 0, 0.8002282]]
dect: [[135, 54, 79, 83, 0, 0.8237426]]
dect: [[125, 39, 92, 110, 0, 0.7744358]]
dect: [[121, 38, 99, 110, 0, 0.7160898]]
dect: [[110, 28, 85, 77, 0, 0.7160898]]

```

5.4 Результаты эксперимента

Подключите модуль K210 к компьютеру с помощью кабеля microUSB. В среде CanMV IDE нажмите кнопку Connect, затем Run, чтобы запустить программу. Также возможно сохранить код как main.py и запускать его автономно на модуле.

После завершения инициализации на LCD-дисплее появится изображение с камеры. При попадании лица в кадр оно будет выделено зелёной рамкой, а в нижнем терминале IDE выводится информация о распознанном объекте.

Вот что тебе нужно сделать **по шагам**, чтобы эксперимент с распознаванием лиц на K210 реально заработал:

1. **Проверь прошивку (важно)**

* Убедись, что на модуле стоит ** заводская прошивка CanMV**, где есть поддержка KPU/AI.

* Если ставил другую прошивку — **прошиваешь обратно заводскую** (иначе скрипт может не видеть KPU или модель).

2. **Подготовь SD-карту**

* Возьми microSD (желательно 8–32 ГБ) и отформатирай в **FAT32**.

* Вставь SD-карту в компьютер (через кардридер).

3. **Скопирай модель на SD-карту**

* На SD-карте создай папки (если их нет):

* `KPU`

* внутри неё: `yolo_face_detect`

* В итоге путь должен быть ровно такой:

* `/sd/KPU/yolo_face_detect/yolo_face_detect.kmodel`

* Скопирай файл модели **yolo_face_detect.kmodel** в эту папку.

4. **Вставь SD-карту в K210**

* Вставляешь microSD в слот на модуле K210 **до запуска кода**.

5. **Подключи модуль к компьютеру**

* Подключи K210 по **microUSB data** (именно data-кабель, не только зарядка).

* Открой **CanMV IDE**.

6. **Подключись в IDE**

* Нажми **Connect**.

* Убедись, что внизу в терминале нет ошибок подключения, и плата определилась.

7. **Открой и запусти пример**

* Открой файл `CanMV\05-AI\yolo_face_detect.py` (или вставь код в редактор).

* Нажми **Run**.

8. **Проверь, что картинка пошла**

* На LCD должна появиться картинка с камеры и надпись FPS.

* Наведи камеру на лицо.

9. **Проверь результат распознавания**

* Если лицо найдено:

* на экране появится **зелёная рамка** вокруг лица

* в терминале будет печать типа `dect: [...]` (координаты прямоугольника)

10. **Если не распознаёт — быстрые действия**

* Поднеси лицо ближе, сделай освещение ярче.

* Понизь порог:

* было `threshold=0.7`

* попробуй `0.5` или `0.4` (будет находить чаще, но возможны ложные срабатывания)

* Если ошибка “не найден файл модели”:

* значит неверный путь или SD не читается → перепроверь пункт 3.

KPU — что это вообще

KPU в K210 — это аппаратный ускоритель нейросетей.

Он:

принимает картинку

прогоняет её через модель (.kmodel)

возвращает «сырые» результаты

KPU не понимает, что такое лицо.

Это уже зашито в модели, а код лишь правильно:

подаёт картинку

расшифровывает выход

Загрузка модели

```
kpu = KPU()
```

```
kpu.load_kmodel("/sd/KPU/yolo_face_detect/yolo_face_detect.kmodel")
```

Anchor — самое непонятное, но простое

```
anchor = (  
    0.893, 1.463, 0.245, 0.389, 1.55, 2.58,  
    0.375, 0.594, 3.099, 5.038, 0.057, 0.090,  
    0.567, 0.904, 0.101, 0.160, 0.159, 0.255  
)
```

Anchor — это “заготовки прямоугольников”, которые YOLO примеряет к картинке.

YOLO не ищет лицо «с нуля».

Он:

накладывает много прямоугольников разного размера

роверяет: похоже ли тут на лицо?

Эти числа — ширина и высота таких заготовок.

Anchors нельзя менять просто так — они подбираются под конкретную модель.

init_yolo2 — главный момент

```
kpu.init_yolo2(  
    anchor,  
    anchor_num=9,  
    img_w=320,  
    img_h=240,  
    net_w=320,  
    net_h=256,  
    layer_w=10,  
    layer_h=8,  
    threshold=0.7,  
    nms_value=0.3,  
    classes=1  
)
```

img_w=320, img_h=240

sensor.set_framesize(sensor.QVGA) # 320x240

net_w=320, net_h=256

Размер входа нейросети

Почему не 320×240?

Потому что:

нейросеть любит размеры кратные 32

256 — ближайшее подходящее значение

Картинка растягивается/дополняется до этого размера перед анализом.

layer_w=10, layer_h=8

Это сетка YOLO.

По-человечески:

YOLO делит изображение на клетки

каждая клетка «смотрит», есть ли в ней объект

Здесь:

10 клеток по ширине

8 по высоте

Эти числа жёстко связаны с моделью, менять нельзя.

classes=1

колько типов объектов модель знает.

1 → только лицо

если бы было:

2 → например, лицо + шлем

threshold=0.7

Это порог уверенности.

0.7 = модель уверена минимум на 70%

меньше → больше находок, но больше ошибок

больше → точнее, но может «не видеть» лица

Для экспериментов:

яркий свет → 0.6–0.7

плохой свет → 0.4–0.5

nms_value=0.3

Убирает дубликаты

YOLO может найти одно лицо несколькими рамками.

NMS:

оставляет лучшую
выкидывает пересекающиеся
Чем меньше значение:
агрессивнее удаляет лишние рамки

Обычно:

0.3 — нормально

трагать редко нужно

od_img — зачем второй буфер

od_img = image.Image(size=(320,256))

KPU не работает напрямую с sensor.snapshot().

Процесс такой:

камера → img
копия в od_img
pix_to_ai() → перевод в формат KPU
KPU считает

Это чисто технический момент K210.

pix_to_ai — магия формата

od_img.pix_to_ai()
Переводит изображение:
из RGB в формат, понятный нейросети

Без этой строки ничего не будет работать.

run_with_output — запуск нейросети

kpu.run_with_output(od_img)
запускает модель

сохраняет результат внутри KPU

regionlayer_yolo2 — получение лиц

dect = kpu.regionlayer_yolo2()

Результат:

[x, y, w, h, confidence, class_id]

Ты используешь:

x, y, w, h

И рисуешь рамку.

Почему рамка иногда «прыгает»

Это нормально:

YOLO работает кадрово

небольшие изменения освещения → другие anchors

лечится увеличением threshold

Главное, что нужно запомнить

Трогать можно безопасно:

threshold

иногда nms_value

НЕЛЬЗЯ менять без другой модели:

anchors

layer_w / layer_h

net_w / net_h

classes

```

# Импорт модулей для работы с камерой, изображениями, временем и LCD-дисплеем
import sensor, image, time, lcd

# Импорт модуля KPU (аппаратный ускоритель нейросетей K210)
from maix import KPU

# ----- ИНИЦИАЛИЗАЦИЯ ОБОРУДОВАНИЯ -----

# Инициализация LCD-дисплея
lcd.init()

# Сброс и инициализация камеры
sensor.reset()

# Установка цветового формата изображения (RGB565 — стандарт для KPU)
sensor.set_pixformat(sensor.RGB565)

# Установка разрешения камеры QVGA (320x240)
sensor.set_framesize(sensor.QVGA)

# Пропуск первых кадров для стабилизации экспозиции и баланса белого
sensor.skip_frames(time = 100)

# Таймер для подсчёта FPS (кадров в секунду)
clock = time.clock()

# ----- ПОДГОТОВКА ИЗОБРАЖЕНИЯ ДЛЯ КПУ -----

# Создание дополнительного изображения для нейросети
# Размер 320x256 используется, так как вход нейросети должен быть кратен 32
od_img = image.Image(size=(320,256))

# ----- НАСТРОЙКА YOLO И КПУ -----

# Anchor-блоки — заранее подобранные размеры прямоугольников,
# которые YOLO использует для поиска объектов (лиц)
anchor = (
    0.893, 1.463,
    0.245, 0.389,
    1.55, 2.58,
    0.375, 0.594,
    3.099, 5.038,
    0.057, 0.090,
    0.567, 0.904,
    0.101, 0.160,
    0.159, 0.255
)

# Создание объекта KPU
kpu = KPU()

# Загрузка нейросетевой модели с SD-карты
kpu.load_kmodel("/sd/KPU/yolo_face_detect/yolo_face_detect.kmodel")

# Инициализация алгоритма YOLOv2
kpu.init_yolo2(
    anchor,          # anchor-блоки
    anchor_num=9,    # количество anchor-блоков
    img_w=320,       # ширина изображения с камеры
    img_h=240,       # высота изображения с камеры
    net_w=320,       # ширина входа нейросети
    net_h=256,       # высота входа нейросети
    layer_w=10,      # количество ячеек YOLO по ширине
    layer_h=8,       # количество ячеек YOLO по высоте
    threshold=0.7,   # порог уверенности распознавания
    nms_value=0.3,   # подавление пересекающихся рамок
    classes=1        # количество классов (1 — лицо)
)

```

```
#  
----- ОСНОВНОЙ ЦИКЛ РАБОТЫ -----  
  
while True:  
    # Обновление таймера (для расчёта FPS)  
    clock.tick()  
  
    # Получение кадра с камеры  
    img = sensor.snapshot()  
  
    # Копирование изображения камеры в буфер нейросети  
    od_img.draw_image(img, 0, 0)  
  
    # Преобразование изображения в формат, понятный KPU  
    od_img.pix_to_ai()  
  
    # Запуск нейросети на текущем кадре  
    kpu.run_with_output(od_img)  
  
    # Получение результатов распознавания YOLO  
    # Каждый элемент содержит координаты и уверенность  
    dect = kpu.regionlayer_yolo2()  
  
    # Получение текущего FPS  
    fps = clock.fps()  
  
    # Если объекты (лица) обнаружены  
    if len(dect) > 0:  
        # Вывод информации о найденных лицах в терминал  
        print("dect:", dect)  
  
        # Отрисовка рамки вокруг каждого найденного лица  
        for l in dect:  
            img.draw_rectangle(  
                l[0], l[1], l[2], l[3],  
                color=(0, 255, 0)  
            )  
  
    # Отображение FPS в левом верхнем углу экрана  
    img.draw_string(  
        0, 0,  
        "%2.1ffps" % fps,  
        color=(0, 60, 128),  
        scale=2.0  
    )  
  
    # Вывод изображения на LCD-дисплей  
    lcd.display(img)  
  
# ----- ОСВОБОЖДЕНИЕ РЕСУРСОВ -----  
  
# Остановка и освобождение KPU (обычно вызывается при выходе из программы)  
kpu.deinit()
```

6 Определение ключевых точек лица (Face feature detection, 68 landmarks)

6.1 Цели эксперимента

В этом эксперименте изучается распознавание лиц и определение ключевых точек лица (landmarks). Камера получает изображение, далее:

первая модель находит лицо и рисует рамку;

вторая модель по области лица вычисляет 68 характерных точек (глаза, брови, нос, рот, контур лица) и отображает их точками, а также печатает информацию в терминал.

Путь к эталонному коду:

CanMV\05-AI\face_detect_68lm.py

6.2 Подготовка к эксперименту

Перед началом работы скопируйте файлы моделей на SD-карту и вставьте SD-карту в слот модуля K210.

6 Обнаружение признаков лица

6.1 Цели эксперимента

В данном уроке изучается функция обнаружения лица. Выполняется анализ изображения, полученного с камеры, сравнение с моделью, выделение лица рамкой, отображение характерных точек лица в виде точек, а также вывод соответствующей информации.

Путь к эталонному коду для данного эксперимента:

CanMV\05-AI\face_detect_68lm.py

6.2 Подготовка перед экспериментом

Сначала необходимо скопировать файл модели на карту памяти, затем вставить карту памяти в слот карты памяти модуля K210.

6.3 Ход эксперимента

Заводская прошивка модуля уже содержит интегрированный модуль алгоритмов AI-зрения. Если ранее была установлена другая прошивка, перед выполнением эксперимента необходимо перепрошить модуль обратно на заводскую прошивку.

Импорт библиотек и инициализация камеры и LCD-дисплея:

```
import sensor, image, time, lcd
from maix import KPU

lcd.init()
sensor.reset()
sensor.set_pixformat(sensor.RGB565)
sensor.set_framesize(sensor.QVGA)
sensor.skip_frames(time = 100)
clock = time.clock()
```

Инициализация параметров, связанных с КРУ.

Для работы КРУ необходимо загрузить файл модели (kmodel).

Путь к файлу модели, используемому в данном эксперименте:

/sd/KPU/yolo_face_detect/face_detect_320x240.kmodel

Для определения соответствия модели используется алгоритм YOLOv2.

```
anchor = (0.1075, 0.126875, 0.126875, 0.175, 0.1465625, 0.2246875,
          0.1953125, 0.25375, 0.2440625, 0.351875, 0.341875, 0.4721875,
          0.5078125, 0.6696875, 0.8984375, 1.099687, 2.129062, 2.425937)
```

```
kpu = KPU()
```

```
kpu.load_kmodel("/sd/KPU/yolo_face_detect/face_detect_320x240.kmodel")
```

```
kpu.init_yolo2(anchor, anchor_num=9, img_w=320, img_h=240,
                net_w=320, net_h=240, layer_w=10, layer_h=8,
                threshold=0.7, nms_value=0.2, classes=1)
```

Инициализация модели КРУ для определения ключевых точек лица.

Путь к файлу модели:

/sd/KPU/face_detect_with_68landmark/landmark68.kmodel

Функция извлечения информации об обнаруженном лице:

```
lm68_kpu = KPU()
```

```
print("ready load model")
```

```
lm68_kpu.load_kmodel("/sd/KPU/face_detect_with_68landmark/landmark68.kmodel")
```

Функция извлечения информации об обнаруженном лице:

```
def extend_box(x, y, w, h, scale):
```

```
    x1_t = x - scale*w
```

```
    x2_t = x + w + scale*w
```

```
    y1_t = y - scale*h
```

```
    y2_t = y + h + scale*h
```

```
    x1 = int(x1_t) if x1_t > 1 else 1
```

```
    x2 = int(x2_t) if x2_t < 320 else 319
```

```
    y1 = int(y1_t) if y1_t > 1 else 1
```

```
    y2 = int(y2_t) if y2_t < 240 else 239
```

```
    cut_img_w = x2 - x1 + 1
```

```
    cut_img_h = y2 - y1 + 1
```

```
    return x1, y1, cut_img_w, cut_img_h
```

Создание бесконечного цикла while, в котором изображение передаётся в КРУ для вычислений.

Сначала выполняется обнаружение лица с помощью нейронной сети YOLOv2,

затем извлекаются координаты лица, которые передаются во вторую модель КРУ

для определения ключевых точек лица. Полученные точки отображаются на экране в виде символов.

```

while True:
    clock.tick()
    img = sensor.snapshot()
    kpu.run_with_output(img)
    dect = kpu.regionlayer_yolo2()
    fps = clock.fps()
    if len(dect) > 0:
        print("dect:",dect)
        for l in dect :
            x1, y1, cut_img_w, cut_img_h = extend_box(l[0], l[1], l[2], l[3], scale=0.08)
            face_cut = img.cut(x1, y1, cut_img_w, cut_img_h)
            a = img.draw_rectangle(l[0],l[1],l[2],l[3], color=(0, 255, 0))
            face_cut_128 = face_cut.resize(128, 128)
            face_cut_128.pix_to_ai()
            out = lm68_kpu.run_with_output(face_cut_128, getlist=True)
            for j in range(68):
                x = int(KPU.sigmoid(out[2 * j])*cut_img_w + x1)
                y = int(KPU.sigmoid(out[2 * j + 1])*cut_img_h + y1)
                a = img.draw_circle(x, y, 2, color=(0, 0, 255), fill=True)
            del (face_cut_128)
            del (face_cut)

    a = img.draw_string(0, 0, "%2.1ffps" %(fps), color=(0, 60, 255), scale=2.0)
    lcd.display(img)

```

6.4 Результаты эксперимента

Подключите модуль K210 к компьютеру с помощью кабеля micro-USB.

В среде CanMV IDE нажмите кнопку Connect, после успешного подключения нажмите Run для запуска примера.

Также можно сохранить код как main.py и запустить его непосредственно на модуле K210.

После завершения инициализации системы на LCD-экране появится изображение с камеры. При наведении камеры на лицо, после его обнаружения, на экране появится зелёная рамка, выделяющая лицо, а контуры органов лица будут отмечены символами. Соответствующие данные будут выведены в окне IDE.



6.5 Итоги эксперимента

Для обнаружения лица требуется карта памяти с загруженными файлами моделей, поэтому необходимо заранее скопировать модели на карту памяти и вставить её в слот модуля K210. Если файл модели не может быть прочитан, система выдаст ошибку.

В настоящее время порог обнаружения лиц установлен как threshold = 0.7. При необходимости повышения точности обнаружения это значение можно скорректировать.

Определение органов лица возможно только после предварительного обнаружения лица. Сначала определяется положение лица, затем изображение лица передаётся в КРУ для повторного вычисления, в результате чего получаются координаты характерных точек лица, после чего контуры органов лица могут быть отображены на экране.

7 Обнаружение маски

7.1 Цель эксперимента

В данном уроке изучается функция обнаружения маски на лице.

Программа анализирует изображение, полученное с камеры, сравнивает его с обученной моделью нейросети и определяет, надета ли маска на лицо. Результат выводится на экран и в терминал.

Путь к эталонному коду данного эксперимента:

CanMV\05-AI\face_mask_detect.py

7.2 Подготовка перед экспериментом

Перед началом эксперимента необходимо:

Скопировать файл модели нейросети на карту памяти

Вставить карту памяти в слот модуля K210

Без карты памяти эксперимент выполнен быть не может, так как модели загружаются не во Flash, а с SD-карты.

/sd/KPU/face_mask_detect/detect_5.kmodel

7.3 Ход эксперимента

Заводская прошивка модуля уже содержит интегрированный модуль алгоритмов компьютерного зрения.

Если ранее была установлена другая прошивка, перед началом эксперимента необходимо перепрошить модуль заводской прошивкой.

Импорт библиотек и инициализация камеры и LCD

```
import sensor, image, time, lcd  
from maix import KPU
```

Используемые модули:

sensor — работа с камерой

image — обработка изображений

time — таймер и расчёт FPS

lcd — вывод изображения на дисплей

KPU — аппаратный ускоритель нейросетей K210

Инициализация LCD-дисплея

Сброс и настройка камеры

Формат изображения: RGB565

Разрешение: 320×240 (QVGA)

Пропуск первых кадров для стабилизации изображения

clock используется для расчёта FPS

Инициализация KPU и модели обнаружения маски

Для работы KPU необходимо загрузить файл модели (.kmodel).

Путь к модели, используемой в данном эксперименте:

/sd/KPU/face_mask_detect/detect_5.kmodel

Для обнаружения используется алгоритм YOLOv2.

od_img = image.Image(size=(320,256), copy_to_fb=False)

od_img — промежуточное изображение, которое:

имеет размер, соответствующий входу нейросети

используется для передачи изображения камеры в KPU

Обратите внимание:

размер входа нейросети (320×256) отличается от размера изображения камеры (320×240).

```
lcd.init()  
sensor.reset()  
sensor.set_pixformat(sensor.RGB565)  
sensor.set_framesize(sensor.QVGA)  
sensor.skip_frames(time = 1000)  
clock = time.clock()
```

```
anchor = (  
    0.156250, 0.222548,  
    0.361328, 0.489583,  
    0.781250, 0.983133,  
    1.621094, 1.964286,  
    3.574219, 3.94000  
)
```

```
kpu = KPU()
print("ready load model")
kpu.load_kmodel("/sd/KPU/face_mask_detect/detect_5.kmodel")
```

Создаётся объект КРУ и загружается модель обнаружения маски.

```
kpu.init_yolo2(
    anchor,
    anchor_num=5,
    img_w=320,
    img_h=240,
    net_w=320,
    net_h=256,
    layer_w=10,
    layer_h=8,
    threshold=0.7,
    nms_value=0.4,
    classes=2
)
```

Параметры:

threshold — порог уверенности обнаружения
nms_value — подавление пересекающихся рамок
classes=2 — два класса:
лицо с маской
лицо без маски

Основной цикл программы

while True:

Программа работает в бесконечном цикле и обрабатывает каждый кадр камеры.

```
img = sensor.snapshot()
od_img.draw_image(img, 0, 0)
od_img.pix_to_ai()
```

Камера делает снимок

Изображение копируется в od_img
Выполняется подготовка изображения для КРУ

<code>kpu.run_with_output(od_img)</code>	Изображение передаётся в нейросеть
<code>dect = kpu.regionlayer_yolo2()</code>	Получается список обнаруженных объектов
<code>fps = clock.fps()</code>	Вычисляется FPS

Анализ результата и отображение

```
if len(dect) > 0:
```

Если объекты обнаружены:

```
if l[4]:
```

l[4] == 1 → лицо в маске

l[4] == 0 → лицо без маски

Лицо в маске:

зелёная рамка

надпись "with mask"

Лицо без маски:

красная рамка

надпись "without mask"

```
lcd.display(img)
```

Итоговое изображение выводится на LCD-дисплей.

7.4 Результаты эксперимента

Подключите модуль K210 к компьютеру с помощью кабеля micro-USB.

В среде CanMV IDE нажмите Connect, затем Run.

После завершения инициализации:

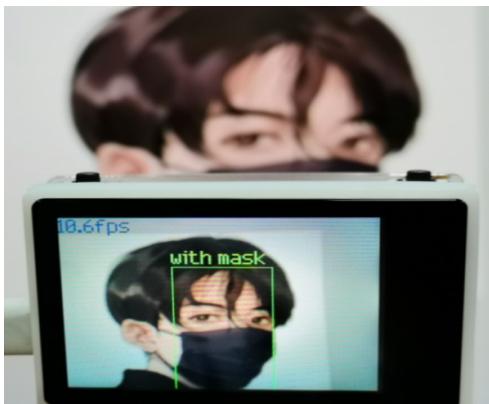
на LCD отображается изображение с камеры

при наличии маски лицо выделяется зелёной рамкой

при отсутствии маски — красной рамкой

результат также выводится в терминал IDE

Примеры работы системы показаны на изображениях:



```
dect: [[121, 76, 73, 88, 0, 0.9914771]]  
dect: [[121, 77, 73, 88, 0, 0.9915406]]  
dect: [[121, 72, 72, 88, 0, 0.9890528]]  
dect: [[121, 77, 73, 88, 0, 0.9930605]]  
dect: [[121, 77, 73, 88, 0, 0.986703]]  
dect: [[121, 76, 73, 88, 0, 0.9868907]]  
dect: [[121, 72, 73, 88, 0, 0.9872982]]  
dect: [[121, 77, 73, 88, 0, 0.9894311]]  
dect: [[121, 76, 73, 88, 0, 0.9894311]]  
dect: [[109, 61, 82, 90, 0, 0.974972]]
```

7.5 Итоги эксперимента

Для обнаружения маски обязательно требуется карта памяти с файлом модели
Если модель не может быть загружена, программа завершится с ошибкой
Порог обнаружения установлен как threshold = 0.7

Для повышения или понижения чувствительности можно корректировать значение threshold

Обнаружение маски реализовано в один этап (одна нейросеть),
в отличие от:

обнаружения лица
обнаружения черт лица (двухэтапная схема)

8 Распознавание лиц (Face Recognition)

8.1 Цель эксперимента

В данном эксперименте изучается функция распознавания лиц.
В отличие от простого обнаружения лица, распознавание позволяет:

сравнивать лицо с ранее сохранёнными лицами
определять, является ли лицо знакомым
выводить оценку похожести (score)

Именно этот пример демонстрирует полноценную систему face recognition, а не просто детекцию.

Путь к эталонному коду эксперимента:
`CanMV\05-AI\face_recog.py`

8.2 Подготовка перед экспериментом

Перед началом эксперимента необходимо:
Скопировать все файлы моделей на карту памяти
Вставить карту памяти в слот модуля K210

Распознавание лиц невозможно без SD-карты, так как используются несколько нейросетевых моделей.

8.3 Ход эксперимента

Заводская прошивка модуля уже содержит модуль AI-зрения.
Если ранее была установлена другая прошивка, необходимо перепрошить модуль заводской версией.

Инициализация камеры и LCD

```
lcd.init()  
sensor.reset()  
sensor.set_pixformat(sensor.RGB565)  
sensor.set_framesize(sensor.QVGA)  
sensor.skip_frames(time = 100)  
clock = time.clock()
```

Камера работает с разрешением 320×240
clock используется для расчёта FPS

Подготовка изображения для извлечения признаков

```
feature_img = image.lImage(size=(64,64), copy_to_fb=False)  
feature_img.pix_to_ai()
```

Создаётся изображение 64×64, в которое будет помещено выровненное лицо.
Это стандартный размер входа для модели извлечения признаков.

Эталонные точки лица (нормализация)

```
FACE_PIC_SIZE = 64  
dst_point = [ ... ]
```

Здесь задаются эталонные координаты 5 ключевых точек лица:
левый глаз
правый глаз
нос
левый угол рта
правый угол рта

Эти точки используются для геометрического выравнивания лица, чтобы:
все лица были повернуты одинаково
масштаб и наклон не влияли на распознавание

Модель обнаружения лица (YOLO)

```
kpu.load_kmodel("/sd/KPU/yolo_face_detect/face_detect_320x240.kmodel")
```

Первая нейросеть:
ищет лицо
возвращает координаты прямоугольника

Без этого шага распознавание невозможно.

Модель определения 5 ключевых точек (LD5)

```
ld5_kpu.load_kmodel("/sd/KPU/face_recognition/ld5.kmodel")
```

Вторая нейросеть:

определяет 5 ключевых точек лица

используется для выравнивания изображения

Это промежуточный этап, который сильно повышает точность распознавания.

Модель извлечения признаков лица

```
fea_kpu.load_kmodel("/sd/KPU/face_recognition/feature_extraction.kmodel")
```

Третья нейросеть:

превращает лицо в вектор признаков

этот вектор уникален для каждого человека

Именно этот вектор используется для сравнения лиц.

Кнопка BOOT — регистрация лица

press boot key to regist face

Нажатие кнопки BOOT сохраняет текущее лицо

Лицо добавляется в список известных лиц

Поэтому код нельзя запускать в CanMV IDE,
кнопка BOOT там не поддерживается.

Переменные распознавания

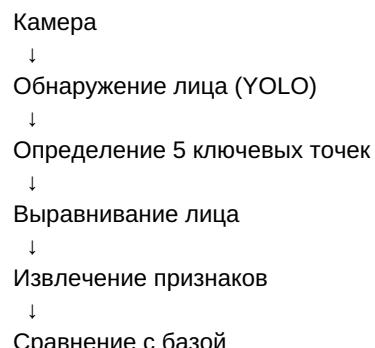
```
record_ftrs = [] # Список сохранённых лиц  
THRESHOLD = 80.5 # Порог распознавания  
recog_flag = False # Флаг распознавания
```

record_ftrs — база известных лиц

THRESHOLD — минимальный score для признания лица знакомым

Основной цикл работы

Полная логика распознавания



Сравнение лиц

```
score = kpu.feature_compare(record_ftrs[j], feature)
```

Каждое новое лицо сравнивается с сохранёнными
Чем выше score, тем лица более похожи

if max_score > THRESHOLD:

Если score выше порога → лицо распознано

Иначе → лицо неизвестно

Отображение результата

Зелёная рамка — лицо распознано

Белая рамка — лицо не зарегистрировано

Отображается:

номер человека (person:0, person:1, ...)

значение score

8.4 Результаты эксперимента

Важно:

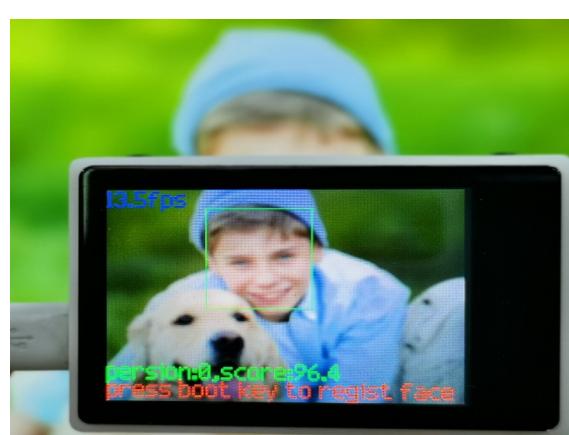
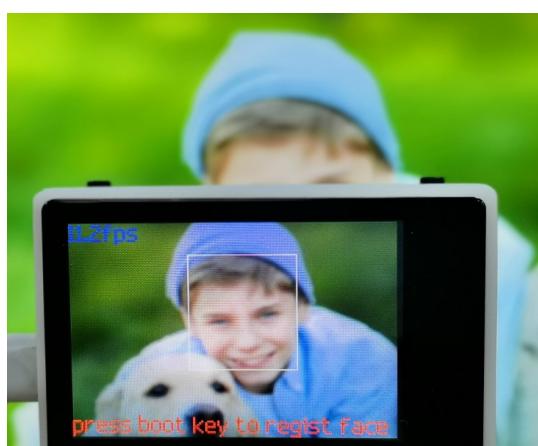
Из-за использования кнопки BOOT код нельзя запускать напрямую из CanMV IDE.

Порядок запуска:

Сохранить код как main.py

Скопировать на K210

Нажать кнопку Reset



Поведение системы

Неизвестное лицо → белая рамка

Нажатие BOOT → регистрация лица

После регистрации → зелёная рамка + score

8.5 Итоги эксперимента

Распознавание лиц использует 3 нейросети
Требуется карта памяти с моделями
Нельзя запускать из CanMV IDE
Лица нумеруются в порядке регистрации:

первый — person:0
второй — person:1 и т.д.

Главное, что нужно понять

Распознавание лица — это не сравнение картинок
Это сравнение векторов признаков
КРУ делает вычисления
К210 управляет процессом

9 Обнаружение объектов (Object Detection)

9.1 Цель эксперимента

В данном уроке рассматривается функция обнаружения объектов.
Программа анализирует изображение, полученное с камеры, сравнивает его с обученной моделью и, при совпадении с одним из типов объектов, выделяет объект рамкой и отображает его название.

В отличие от распознавания лиц, здесь система:
ищет различные типы объектов
определяет класс объекта
работает по модели с несколькими классами

Путь к эталонному коду эксперимента:
CanMV\05-AI\voc20_object_detect.py

9.2 Подготовка перед экспериментом

Перед началом эксперимента необходимо:
Скопировать файл модели нейросети на карту памяти
Вставить карту памяти в слот модуля K210
Без SD-карты эксперимент выполнен быть не может, так как модель загружается не во Flash, а с карты памяти.

9.3 Ход эксперимента

Заводская прошивка модуля уже содержит модуль алгоритмов AI-зрения.
Если ранее была установлена другая прошивка, перед экспериментом необходимо перепрошить модуль заводской прошивкой.

Инициализация библиотек, камеры и LCD

```
import sensor, image, time, lcd  
from maix import KPU
```

Используемые библиотеки:

sensor — управление камерой

image — обработка изображений

time — таймер и FPS

lcd — вывод изображения

KPU — аппаратный ускоритель нейросетей

```
lcd.init()  
sensor.reset()  
sensor.set_pixformat(sensor.RGB565)  
sensor.set_framesize(sensor.QVGA)  
sensor.skip_frames(time = 100)  
clock = time.clock()
```

Разрешение камеры: 320×240 (QVGA)

Цветовой формат: RGB565

Пропуск начальных кадров для стабилизации изображения

Список распознаваемых объектов

```
obj_name = (  
    "aeroplane", "bicycle", "bird", "boat", "bottle",  
    "bus", "car", "cat", "chair", "cow",  
    "diningtable", "dog", "horse", "motorbike", "person",  
    "pottedplant", "sheep", "sofa", "train", "tvmonitor"  
)
```

Модель поддерживает 20 типов объектов (VOC20):

№Объект

0Самолёт

1Велосипед

2Птица

3Лодка

4Бутылка

5Автобус

6Автомобиль

7Кошка

8Стул

9Корова

10Стол

11Собака

12Лошадь

13Мотоцикл

14Человек

15Растение в горшке

16Овца

17Диван

18Поезд

19Телевизор

Обнаруживаются только эти объекты, остальные игнорируются.

Инициализация KPU и модели обнаружения объектов

```
od_img = image.Image(size=(320,256))
```

Создаётся буфер изображения для нейросети:
размер соответствует входу модели
используется для передачи изображения в KPU

```
anchor = (  
    1.3221, 1.73145,  
    3.19275, 4.00944,  
    5.05587, 8.09892,  
    9.47112, 4.84053,  
    11.2364, 10.0071  
)
```

anchor — опорные размеры YOLO
Жёстко привязаны к обученной модели

```
kpu = KPU()  
print("ready load model")  
kpu.load_kmodel("/sd/KPU/voc20_object_detect/voc20_detect.kmodel")
```

Загрузка модели обнаружения объектов VOC20.

```
kpu.init_yolo2(  
    anchor,  
    anchor_num=5,  
    img_w=320,  
    img_h=240,  
    net_w=320,  
    net_h=256,  
    layer_w=10,  
    layer_h=8,  
    threshold=0.7,  
    nms_value=0.2,  
    classes=20  
)
```

Основные параметры:

threshold = 0.7 — минимальная уверенность обнаружения
classes = 20 — количество классов объектов

Основной цикл работы

```
while True:
```

Программа работает в реальном времени.

Логика работы цикла

Камера

↓

Подготовка изображения

↓

KPU (YOLO)

↓

Определение класса объекта

↓

Отрисовка рамки и названия

```
img = sensor.snapshot()  
od_img.draw_image(img, 0, 0)  
od_img.pix_to_ai()
```

Камера делает снимок

Изображение готовится для KPU

Отображение результатов

```
img.draw_rectangle(l[0], l[1], l[2], l[3], color=(0, 255, 0))  
img.draw_string([l[0], l[1], obj_name[l[4]]], color=(0, 255, 0))
```

Для каждого найденного объекта:

рисуется зелёная рамка

отображается название объекта

Также выводится FPS.

9.4 Результаты эксперимента

После запуска программы:

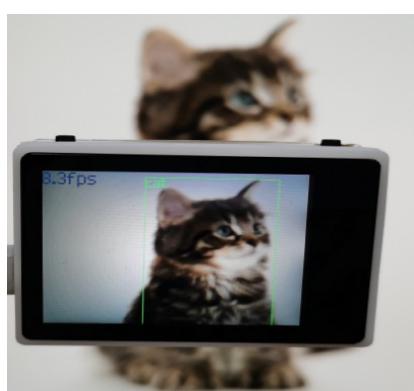
LCD отображает изображение с камеры

Обнаруженные объекты выделяются рамкой

Название объекта выводится рядом

В терминале IDE печатается информация о детекции

Примеры работы показаны на изображениях:



```
dect: [[156, 4, 161, 89, 6, 0.7398304]]  
dect: [[157, 4, 162, 90, 6, 0.7839438]]  
dect: [[157, 5, 161, 89, 6, 0.616123]]  
dect: [[157, 4, 161, 89, 6, 0.6336891]]  
dect: [[157, 4, 161, 90, 6, 0.6782631]]  
dect: [[157, 4, 162, 90, 6, 0.8122816]]  
dect: [[157, 5, 161, 89, 6, 0.6923399]]  
dect: [[156, 4, 161, 90, 6, 0.69679]]  
dect: [[157, 9, 161, 81, 6, 0.693275]]  
dect: [[156, 4, 161, 89, 6, 0.7675117]]  
dect: [[157, 4, 161, 90, 6, 0.7341752]]
```

9.5 Итоги эксперимента

Для обнаружения объектов требуется SD-карта с моделью
Если модель не загружается — программа завершится с ошибкой
Чувствительность определяется параметром threshold

Обнаруживаются только объекты из списка obj_name

Главное, что нужно понять

Обнаружение объектов — это классификация + локализация
КРУ выполняет все вычисления нейросети
CPU управляет логикой и выводом
Скорость работы — до 15–20 FPS

10 Самообучающаяся классификация (Self-learning Classification)

10.1 Цель эксперимента

В данном уроке изучается функция самообучающейся классификации микроконтроллера K210.

В ходе эксперимента:

выбираются три различных объекта

каждый объект фотографируется с разных ракурсов

K210 сам обучается различать эти объекты

далее камера определяет, к какому классу относится текущий объект

Важно:

это не заранее обученная модель, а пример локального обучения прямо на устройстве.

Путь к эталонному коду эксперимента:

CanMV\05-AI\self_learning.py

10.2 Подготовка перед экспериментом

Перед началом эксперимента необходимо:

Скопировать файл модели на карту памяти

Вставить карту памяти в слот модуля K210

Без SD-карты эксперимент невозможен, так как модель загружается с карты памяти.

10.3 Ход эксперимента

Заводская прошивка модуля уже содержит модуль AI-зрения.

Если ранее была установлена другая прошивка, перед экспериментом необходимо перепрошить модуль заводской прошивкой.

Загрузка библиотек и модели

```
kpu = KPU()
print("ready load model")
kpu.load_kmodel("/sd/KPU/self_learn_classifier(mb-0.25.kmodel")
```

Используется модель: mb-0.25.kmodel

Эта модель не классифицирует сама по себе, она используется для извлечения признаков, на основе которых затем выполняется обучение.

Архитектура программы: машина состояний

Весь алгоритм построен как машина состояний (state machine) и разделён на три основных режима:

Состояние	Назначение
INIT	Инициализация и подсказки
TRAIN	Сбор изображений и обучение
CLASSIFY	Классификация объектов

```
if state_machine.current_state == STATE.INIT:
    loop_init()
elif state_machine.current_state == STATE.CLASSIFY:
    loop_classify()
elif state_machine.current_state == STATE.TRAIN_CLASS_1 \
    or state_machine.current_state == STATE.TRAIN_CLASS_2 \
    or state_machine.current_state == STATE.TRAIN_CLASS_3:
    loop_capture()
```

INIT — режим инициализации

Назначение
отображение инструкции
управление кнопкой BOOT

```
def loop_init():
    На экране отображается:
        название демо
        подсказки по управлению
```

Управление:
короткое нажатие BOOT → следующий этап
долгое нажатие BOOT → перезапуск

TRAIN — режим обучения (сбор данных)

Назначение
сбор изображений объектов
формирование классов

`def loop_capture():`

В этом режиме:
камера показывает изображение

на экране отображается:
номер класса
номер кадра (P1–P5)

По умолчанию:
3 класса
5 изображений на каждый класс

всего 15 изображений

Как проходит обучение

- 1 Показать первый объект
- 2 Нажать ВООТ → сохранить изображение
- 3 Немного изменить угол
- 4 Повторить до 5 раз
- 5 Перейти к следующему объекту

CLASSIFY — режим классификации

Назначение
определение, к какому классу относится текущий объект

`def loop_classify():`

Алгоритм работы:
Камера делает снимок
КРУ извлекает вектор признаков
Вектор сравнивается с сохранёнными признаками
Вычисляется score (оценка похожести)

Выбирается класс с максимальным score

`if high > THRESHOLD:
 bottom_msg = "class:{} score:{:.2f}".format(index + 1, high)`

10.4 Результаты эксперимента

В центре — текущий режим (например, Classification)

Внизу — результат:

class:1 score:85.3

Чем выше score, тем увереннее классификация.

Что отображается на экране

Важно:

Эксперимент использует кнопку BOOT, поэтому нельзя запускать код из CanMV IDE.

Правильный порядок запуска:

Сохранить код как main.py

Скопировать на K210

Нажать кнопку RESET

Этапы работы

1 Экран Self Learning Demo

2 Короткое нажатие BOOT → начало обучения

3 Обучение класса 1 (5 изображений)

4 Обучение класса 2

5 Обучение класса 3

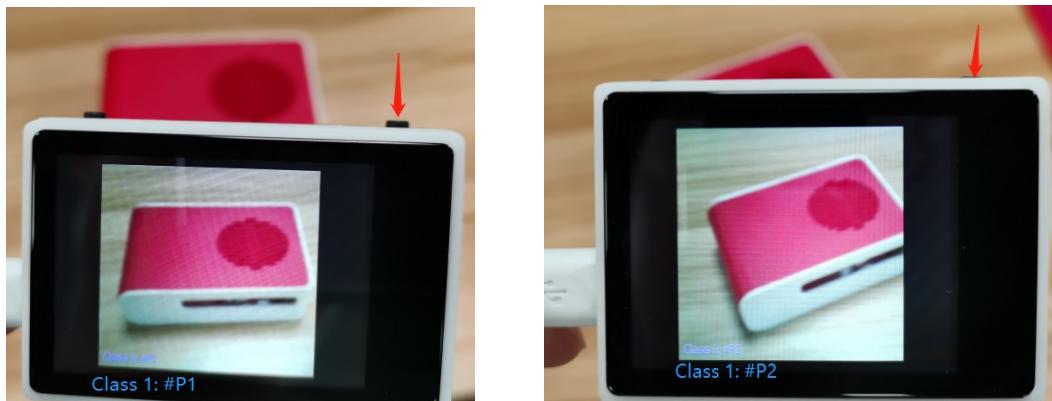
6 Автоматический переход в режим классификации



2 Короткое нажатие BOOT → начало обучения



Обучение класса 1 (5 изображений)



Обучение класса 2 (5 изображений)



Обучение класса 3



Автоматический переход в режим классификации



Результат классификации

Навести камеру на один из обученных объектов

Внизу отображается:

номер класса

score

Если результат плохой:

нажать RESET

или долго удерживать BOOT для перезапуска



10.5 Итоги эксперимента

Самообучение работает без переобучения модели

Используется локальная база признаков

Требуется SD-карта

Нельзя запускать через CanMV IDE

Качество сильно зависит от:

фона освещения

разнообразия ракурсов

Практические рекомендации

Используйте объекты, контрастные фону

Меняйте угол съёмки при обучении

Не используйте слишком похожие объекты

Помните: обучается только лицевая сторона

Обратная сторона объекта часто не распознаётся,
так как не была показана при обучении.

Главное, что нужно понять

Это классификация, а не распознавание объектов VOC

KPU извлекает признаки

CPU сравнивает признаки

Обучение происходит на устройстве, но ограничено

Классификация vs Распознавание — в чём разница?

Коротко (если в одно предложение)

Классификация — что это за объект?

Распознавание — кто или что именно это?

1 Классификация (Classification)

Что делает

Классификация отвечает на вопрос:

К какому классу относится этот объект?

Пример

Ты обучил систему на:

кружка

телефон

пульт

Система видит объект и говорит: Это объект класса 2

Она не знает, что это телефон —

она знает только, что он похож на то, что ты показал как класс 2.

Как работает на K210

Камера снимает объект

KPU извлекает признаки изображения

Признаки сравниваются с теми, что ты сохранил при обучении

Выбирается самый похожий класс

Имена классов задаёшь ты сам (Class 1, Class 2, Class 3)

Ключевые особенности классификации

Можно обучать прямо на устройстве

Не нужен заранее обученный датасет

Классы не имеют смысла вне текущего обучения

Не распознаёт «кто именно», только «на что похоже»

2 Распознавание (Recognition)

Что делает

Распознавание отвечает на вопрос:

Это уже известный объект или человек?

Пример (лица)

Ты сохранил:

лицо Алексея

лицо Марии

Система видит лицо и говорит:

Это Алексей, score = 92.4

или

Неизвестное лицо

Как работает на K210

Камера находит лицо

KPU извлекает уникальный вектор признаков

Вектор сравнивается с базой сохранённых лиц

Если похожесть выше порога — лицо распознано

Каждый человек — уникальный объект, а не класс

Ключевые особенности распознавания

Можно отличать конкретных людей

Используются специализированные модели

Требует точной подготовки (ключевые точки, выравнивание)

Сложнее, чем классификация

3 Обнаружение объектов (для контраста)

Чтобы не путать, добавим третий тип

Обнаружение объектов отвечает на вопрос:

Где объект и что это за тип?

Пример:

person, bottle, dog

Классы жёстко заданы моделью, обучить новые нельзя.

Сравнение в одной таблице

КритерийКлассификацияРаспознавание

Вопрос«На что это похоже?»«Это уже знакомый объект?»

ОбучениеНа устройстведо запуска

КлассыПользовательскиеУникальные объекты

ПримерClass 1 / Class 2Person 0 / Person 1

ПереобучениеБыстроТребует сохранения

СложностьНизкаяВысокая

Тип задачиСравнение признаковИдентификация

Пример из жизни

Классификация

Это один из трёх предметов, которые я показал

Распознавание

Это тот самый предмет, который я видел раньше

Главное, что нужно запомнить

Классификация ≠ распознавание

Классификация — группировка

Распознавание — идентификация

KPU в обоих случаях извлекает признаки

Разница — в логике сравнения

Когда что использовать

Хочешь различать 3–5 предметов быстро → классификация

Хочешь узнавать конкретного человека → распознавание

Хочешь узнать что за объект вообще → object detection

11 Распознавание рукописных цифр

11.1 Цели эксперимента

В данном уроке рассматривается функция распознавания цифр на платформе K210. Модуль способен распознавать как рукописные, так и печатные цифры.

Путь к эталонному коду эксперимента:

CanMV\05-AI\mnist.py

11.2 Подготовка к эксперименту

Сначала скопируйте файл модели на карту памяти, затем вставьте карту памяти в соответствующий слот модуля K210.

11.3 Ход эксперимента

Заводская прошивка модуля уже содержит встроенный модуль алгоритмов компьютерного зрения. Если ранее была загружена другая прошивка, перед началом эксперимента необходимо перепрошить модуль на заводскую прошивку.

Импортируйте необходимые библиотеки, инициализируйте камеру и LCD-дисплей, затем загрузите файл модели:

/sd/KPU/mnist/uint8_mnist_cnn_model.kmodel

```
kpu = KPU()  
kpu.load_kmodel("/sd/KPU/mnist/uint8_mnist_cnn_model.kmodel")
```

Создайте бесконечный цикл while для получения изображения с камеры.

В цикле выполняются следующие операции:

захват изображения;
преобразование в градации серого;
изменение размера до 112×112;
инверсия цветов;
усиление контраста символов;
передача изображения в KPU для вычислений;

запуск нейросетевой модели;

получение наиболее вероятного результата распознавания и его оценки.

```
while True:  
    gc.collect()  
    img = sensor.snapshot()  
    img_mnist1 = img.to_grayscale(1)  
    img_mnist2 = img_mnist1.resize(112,112)  
    img_mnist2.invert()  
    img_mnist2.stretch_char(1)  
    img_mnist2.pix_to_ai()  
  
    out = kpu.run_with_output(img_mnist2, getlist=True)  
    max_mnist = max(out)  
    index_mnist = out.index(max_mnist)  
    score = KPU.sigmoid(max_mnist)
```

Поскольку распознанные цифры могут быть определены неверно, рекомендуется использовать белый фон.

На практике:

полностью белый фон может ошибочно распознаваться как цифра 1;

полностью чёрный фон может ошибочно распознаваться как цифра 5.

Поэтому для цифр 1 и 5 используется отдельная проверка: если коэффициент уверенности (score) больше 0.999, результат считается корректным. В противном случае считается, что цифра не распознана. Значение 0.999 можно изменять в зависимости от качества распознавания.

Распознанная цифра выводится на экран:

```
if index_mnist == 1:  
    if score > 0.999:  
        display_str = "num: %d" % index_mnist  
        print(display_str, score)  
        img.draw_string(4,3,display_str,color=(0,0,0),scale=2)  
    elif index_mnist == 5:  
        if score > 0.999:  
            display_str = "num: %d" % index_mnist  
            print(display_str, score)  
            img.draw_string(4,3,display_str,color=(0,0,0),scale=2)  
    else:  
        display_str = "num: %d" % index_mnist  
        print(display_str, score)  
        img.draw_string(4,3,display_str,color=(0,0,0),scale=2)  
  
lcd.display(img)
```

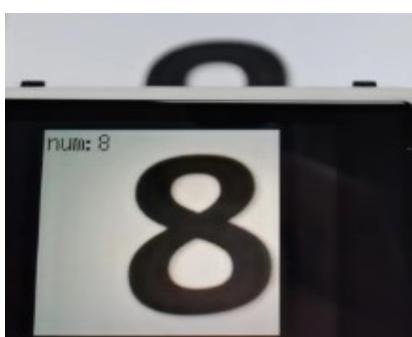
11.4 Результаты эксперимента

Подключите модуль K210 к компьютеру с помощью кабеля microUSB.

В среде CanMV IDE нажмите кнопку Connect, после установления соединения нажмите Run для запуска программы.

Также можно сохранить код под именем main.py и загрузить его напрямую в модуль K210.

После завершения инициализации на LCD-экране появится изображение с камеры. Наведите камеру на рукописную или печатную цифру — в левом верхнем углу экрана отобразится распознанное число.



11.5 Итоги эксперимента

Модуль K210 способен распознавать как рукописные, так и печатные цифры. Так как обучение модели проводилось на чёрных символах на белом фоне, рекомендуется использовать именно такой формат изображения.

Если фон слишком сложный или цифры слишком маленькие, возможны ошибки распознавания.

```

# Импорт необходимых библиотек
import sensor, image, time, lcd      # Работа с камерой, изображениями, временем и дисплеем
from maix import KPU                # KPU — нейропроцессор на чипе Kendryte (ускоряет работу нейросетей)
import gc                            # Модуль для ручной очистки памяти (важно на встраиваемых устройствах)

# === 1. ИНИЦИАЛИЗАЦИЯ ОБОРУДОВАНИЯ ===
lcd.init()                          # Включаем дисплей для вывода результатов
sensor.reset()                      # Перезагружаем камеру (обязательный первый шаг)

# Настройка параметров камеры:
sensor.set_pixformat(sensor.RGB565) # Формат цвета: 16-битный RGB (5 бит на красный, 6 на зелёный, 5 на синий)
                                    # Важно: для чёрно-белых задач (как MNIST) позже преобразуем в градации серого

sensor.set_framesize(sensor.QVGA)    # Размер кадра: 320x240 пикселей (стандартный для встраиваемых систем)
sensor.set_windowing((224, 224))    # Обрезаем центральную область 224x224 пикселей — уменьшаем нагрузку на процессор

sensor.skip_frames(time=100)         # Пропускаем первые кадры (100 мс), чтобы камера успела настроить экспозицию и баланс белого

clock = time.clock()               # Создаём таймер для измерения производительности (не используется в коде, но полезен для отладки)

# === 2. ЗАГРУЗКА НЕЙРОСЕТИ ===
kpu = KPU()                        # Инициализируем нейропроцессор
kpu.load_kmodel("/sd/KPU/mnist/uint8_mnist_cnn_model.kmodel")
                                    # Загружаем предобученную модель для распознавания цифр MNIST
                                    # uint8 — модель использует 8-битные числа для экономии памяти и ускорения
                                    # Архитектура: свёрточная нейросеть (CNN), идеальна для анализа изображений

# === 3. ОСНОВНОЙ ЦИКЛ РАСПОЗНАВАНИЯ ===
while True:
    gc.collect()                    # Очищаем память перед обработкой нового кадра (критично для устройств с ограниченной памятью)

    # === 3.1. ЗАХВАТ И ПРЕДОБРАБОТКА ИЗОБРАЖЕНИЯ ===
    img = sensor.snapshot()          # Делаем снимок с камеры (цветное изображение 224x224)

    img_mnist1 = img.to_grayscale(1) # Преобразуем в чёрно-белое (градации серого)
                                    # Почему? Модель MNIST обучена на монохромных цифрах — цвет не несёт полезной информации

    img_mnist2 = img_mnist1.resize(112, 112) # Уменьшаем размер до 112x112
                                    # Важно: входной слой модели ожидает именно такой размер (проверяйте архитектуру модели!)

    img_mnist2.invert()              # Инвертируем цвета: белый фон → чёрный, чёрная цифра → белая
                                    # Стандартный датасет MNIST: цифры белые на чёрном фоне

    img_mnist2.stretch_char(1)       # "Растягиваем" цифру по краям изображения (автоматически центрирует и нормализует размер символа)
                                    # Улучшает распознавание: цифра занимает всю доступную область, как в обучающих данных

    img_mnist2.pix_to_ai()          # Копируем данные изображения в память нейропроцессора (подготовка к инференсу)

    # === 3.2. ЗАПУСК НЕЙРОСЕТИ (ИНФЕРЕНС) ===
    out = kpu.run_with_output(img_mnist2, getlist=True)
                                    # Запускаем модель на изображении. Результат — список из 10 чисел (логитов)
                                    # Каждое число соответствует "уверенности" сети в цифре от 0 до 9

    max_mnist = max(out)            # Находим максимальное значение в выходном векторе
    index_mnist = out.index(max_mnist) # Определяем индекс этого максимума — это предсказанная цифра (0..9)

    score = KPU.sigmoid(max_mnist)   # Применяем сигмоиду к максимальному логиту для получения "уверенности" в диапазоне [0, 1]
                                    # Примечание: для многоклассовой классификации обычно используется softmax,
                                    # но здесь сигмоида применяется только к максимальному значению как приближение

    # === 3.3. ПОСТОДРАБОТКА И ВИЗУАЛИЗАЦИЯ ===
    # Особая логика для цифр 1 и 5: требуем очень высокую уверенность (>0.999)
    # Почему? Эти цифры часто путаются (например, единица похожа на семёрку, пятёрка — на шестёрку)
    # Это пример "калибровки порогов" в реальных системах для снижения ложных срабатываний
    if index_mnist == 1:
        if score > 0.999:
            display_str = "num: %d" % index_mnist
            print(display_str, score) # Выводим в консоль для отладки
            img.draw_string(4, 3, display_str, color=(0,0,0), scale=2) # Рисуем текст на исходном цветном изображении
    elif index_mnist == 5:
        if score > 0.999:
            display_str = "num: %d" % index_mnist
            print(display_str, score)
            img.draw_string(4, 3, display_str, color=(0,0,0), scale=2)
    else:
        # Для остальных цифр выводим результат без строгой фильтрации
        display_str = "num: %d" % index_mnist
        print(display_str, score)
        img.draw_string(4, 3, display_str, color=(0,0,0), scale=2)

    lcd.display(img)                # Отображаем изображение с наложенным результатом на дисплей

# === 4. ЗАВЕРШЕНИЕ РАБОТЫ (не достигается в бесконечном цикле, но важно для корректного освобождения ресурсов) ===
kpu.deinit()
                                    # Освобождаем ресурсы нейропроцессора

```

Ключевые концепции для начинающих:

Предобработка изображения — критически важный этап. Модель работает только с данными в том формате, на котором обучалась (размер, цвет, ориентация).

Инверсия цветов — не "магия", а соответствие формату обучающего датасета (MNIST: белые цифры на чёрном фоне).

Нормализация символа (stretch_char) — помогает модели игнорировать положение и размер цифры в кадре.

Пороги уверенности — в реальных системах редко используют "голый" результат нейросети. Добавляют фильтрацию по уверенности для повышения надёжности.

Ограничения памяти — вызов gc.collect() показывает, что на встраиваемых устройствах нужно управлять памятью вручную.

Архитектура CNN — свёрточные сети автоматически выделяют признаки (линии, углы, контуры), что делает их идеальными для анализа изображений.

Совет для практики: попробуйте убрать invert() или stretch_char() и понаблюдайте, как упадёт точность распознавания — это наглядно покажет важность предобработки!

12. Распознавание нескольких цветов

1.1 Цели эксперимента

Данный урок посвящён функциям распознавания различных цветов. На основе значений цветового пространства LAB осуществляется детектирование объектов разного цвета (красный, жёлтый, синий, зелёный) с выделением их контуров без необходимости предварительного сбора образцов.

Путь к эталонному коду для данного эксперимента:

CanMV\05-AI\multi_color_recognition.py

1.2 Проведение эксперимента

Заводская прошивка модуля уже включает модуль алгоритмов машинного зрения.

\Если ранее была установлена другая прошивка, перед проведением эксперимента необходимо восстановить заводскую версию.

Импорт необходимых библиотек и инициализация камеры и ЖК-дисплея:

```
import sensor
import image
import time
import lcd

lcd.init()
sensor.reset()
sensor.set_pixformat(sensor.RGB565)
sensor.set_framesize(sensor.QVGA)
sensor.skip_frames(time = 100)
sensor.set_auto_gain(False)
sensor.set_auto_whitebal(False)

clock = time.clock()
```

Задание диапазонов пороговых значений LAB для различных цветов:

```
color_thresholds = [
    (31, 69, 27, 58, 14, 36), # Красный
    (14, 61, -39, -6, 0, 14), # Зелёный
    (14, 66, 1, 38, -56, -12), # Синий
    (49, 77, -8, 52, 16, 60), # Жёлтый
]
color_strings = ['Red', 'Green', 'Blue', 'Yellow']
```

Создание цикла для распознавания цветов на изображениях с камеры и отображения результатов на экране:

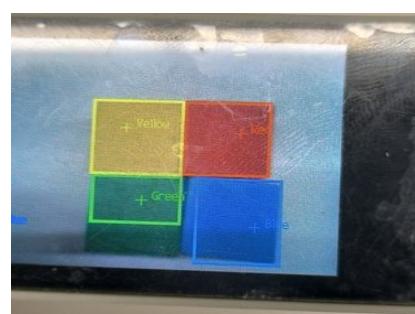
```
while True:
    clock.tick()
    img = sensor.snapshot()
    for color_idx, threshold in enumerate(color_thresholds):
        blobs = img.find_blobs([threshold], pixels_threshold=100, area_threshold=100, merge=True,
margin=10)
        if blobs:
            for blob in blobs:
                color_new = (255, 255, 255)
                if color_idx == 0:
                    color_new = (255, 0, 0)
                elif color_idx == 1:
                    color_new = (0, 255, 0)
                elif color_idx == 2:
                    color_new = (0, 0, 255)
                elif color_idx == 3:
                    color_new = (255, 255, 0)
                img.draw_rectangle(blob.rect(), color=color_new, thickness=3)
                img.draw_cross(blob.cx(), blob.cy(), color=color_new)
                img.draw_string(blob.cx() + 10, blob.cy() - 10, color_strings[color_idx], color=color_new)
    lcd.display(img)
    print(clock.fps())
```

1.3 Результаты эксперимента

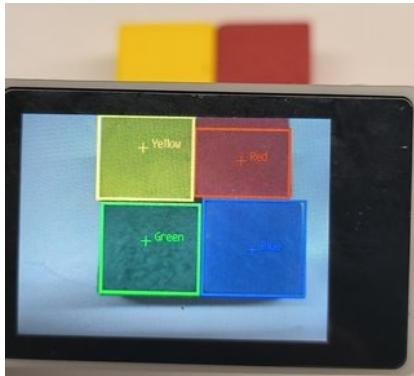
Подключите модуль K210 к компьютеру с помощью кабеля microUSB, нажмите кнопку подключения в среде CanMV IDE. После установления соединения нажмите кнопку запуска для выполнения кода. Также можно сохранить код как main.py и запустить его непосредственно на модуле K210.

После завершения инициализации система отображает изображение с камеры на ЖК-дисплее и начинает распознавание цветов, выводя соответствующие подписи и контуры на экран.

Результат распознавания камерой OV2640:



Результат распознавания камерой GC2145:



1.4 Итоги эксперимента

Основной принцип распознавания нескольких цветов заключается в анализе значений цветового пространства LAB и их сравнении с данными, полученными камерой.

При соответствии пороговым значениям объект выделяется прямоугольником с подписью соответствующего цвета.

Следует учитывать, что результаты распознавания сильно зависят от условий освещения: значения LAB могут изменяться при разных источниках и интенсивности света.

В следующих разделах описано, как скорректировать пороговые значения LAB в зависимости от освещения, а также как добавить собственные цвета помимо указанных в уроке.

2.1 Добавление новых цветов для распознавания

Для распознавания дополнительных цветов необходимо добавить соответствующие пороговые значения LAB и названия в исходный код. Пример модификации:

```
color_thresholds = [
    (31, 69, 27, 58, 14, 36), # Красный
    (14, 61, -39, -6, 0, 14), # Зелёный
    (14, 66, 1, 38, -56, -12), # Синий
    (49, 77, -8, 52, 16, 60), # Жёлтый
    (значения LAB для нового цвета) # Пользовательское значение LAB — требуется
    ручная настройка
]
color_strings = ['Red', 'Green', 'Blue', 'Yellow', 'Название нового цвета'] # Требуется ручная
    модификация
```

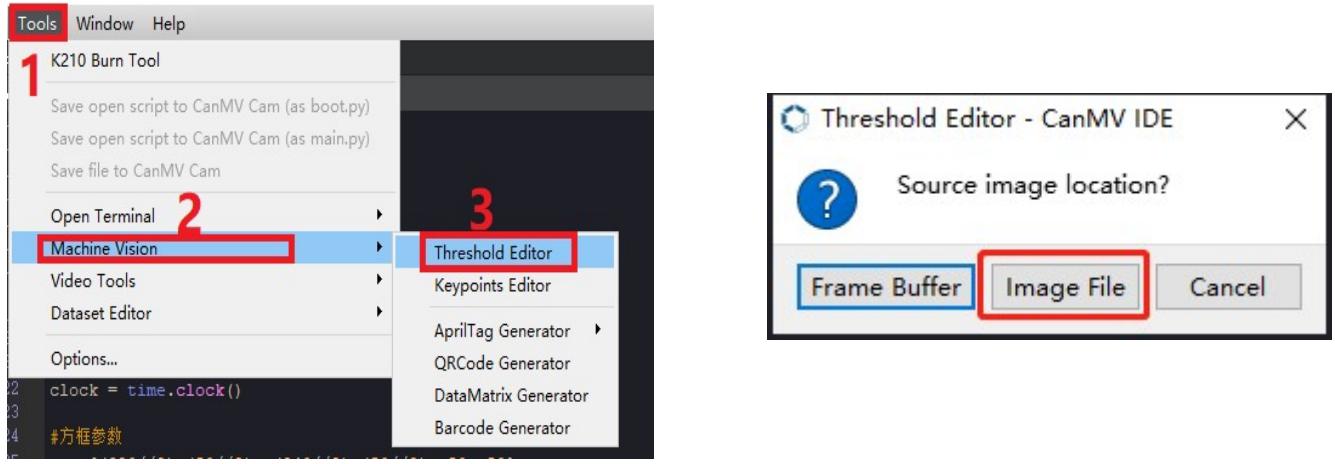
2.2 Получение пороговых значений LAB для новых цветов

Для получения или корректировки пороговых значений LAB под текущие условия освещения можно воспользоваться встроенным инструментом среды CanMV IDE.

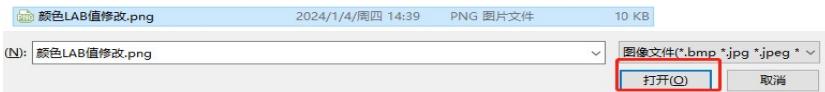
Запустите CanMV IDE и выберите: «Инструменты → Machine Vision → Threshold Editor».

Нажмите «Image File».

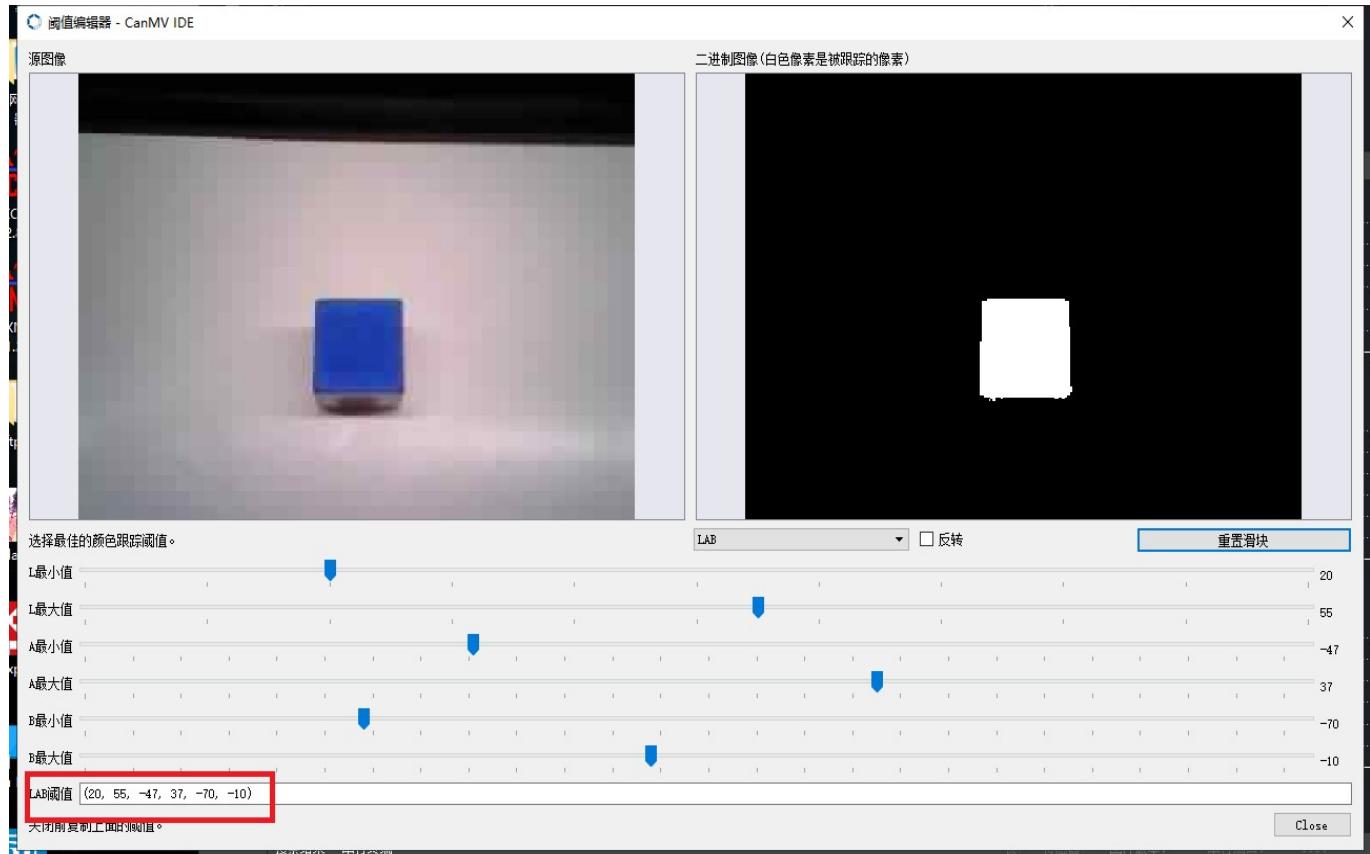
Также доступен вариант с использованием буфера кадров (frame buffer), но в данном руководстве он не рассматривается. Обратите внимание: для этого требуется подключение K210 к CanMV с включённым буфером кадров.



Выберите изображение, содержащее новый цвет или отражающее текущие условия освещения (рекомендуется использовать изображения с чистым фоном и минимальными цветовыми помехами).



Перемещайте ползунки до тех пор, пока область распознаваемого цвета не станет белой, а всё остальное — чёрным. Зафиксируйте полученные пороговые значения LAB.



Вставьте полученные значения в исходный код в соответствии с инструкцией из раздела 2.1.

```

"""
@file: multi_color_recognition.py
@company: yahboom
@description: Распознавание нескольких цветов
"""

import sensor
import image
import time
import lcd

# Инициализация LCD-дисплея
lcd.init()
# Инициализация сенсора камеры
sensor.reset()
sensor.set_pixformat(sensor.RGB565)
sensor.set_framesize(sensor.QVGA)
sensor.skip_frames(time = 100)
sensor.set_auto_gain(False) # Отключаем автонастройку усиления
sensor.set_auto_whitebal(False) # Отключаем автонастройку баланса белого
# Счётчик для измерения частоты кадров
clock = time.clock()

# Параметры прямоугольника (не используется в текущей версии кода)
r = [(320//2)-(50//2), (240//2)-(50//2), 50, 50]

# Пороговые значения LAB для распознавания различных цветов
color_thresholds = [
    (31, 69, 27, 58, 14, 36), # Красный (Red)
    #(20,80,20,62,20,35), # Альтернативный порог для красного

    #(49, 74, -26, 48, 15, 55), # Зелёный (альтернативный вариант)
    (14, 61, -39, -6, 0, 14), # Зелёный (Green)

    (14, 66, 1, 38, -56, -12), # Синий (Blue)
    #(21,50,-7,8,-35,-11), # Альтернативный порог для синего

    (49, 77, -8, 52, 16, 60), # Жёлтый (Yellow)
    #(65, 78, -10, -5, 38, 50), # Альтернативный порог для жёлтого
]

# Названия цветов для отображения (рекомендуется оставлять на английском из-за ограничений шрифтов дисплея)
color_strings = ['Red', 'Green', 'Blue', 'Yellow']

print("Запуск распознавания цветов...")

# Основной цикл программы
while True:
    clock.tick() # Начало отсчёта времени для расчёта FPS
    img = sensor.snapshot() # Захват кадра с камеры

    # Перебор всех заданных цветовых порогов
    for color_idx, threshold in enumerate(color_thresholds):
        # Поиск областей, соответствующих текущему цветовому порогу
        blobs = img.find_blobs([threshold], pixels_threshold=100, area_threshold=100, merge=True, margin=10)

        if blobs: # Если найдены объекты заданного цвета
            for blob in blobs:
                # Определение цвета рамки в RGB в зависимости от индекса распознанного цвета
                color_new = (255, 255, 255)
                if color_idx == 0:
                    color_new = (255, 0, 0) # Красный
                elif color_idx == 1:
                    color_new = (0, 255, 0) # Зелёный
                elif color_idx == 2:
                    color_new = (0, 0, 255) # Синий
                elif color_idx == 3:
                    color_new = (255, 255, 0) # Жёлтый

                # Визуализация результатов распознавания
                img.draw_rectangle(blob.rect(), color=color_new, thickness=3) # Рамка вокруг объекта
                img.draw_cross(blob.cx(), blob.cy(), color=color_new) # Крест в центре объекта
                img.draw_string(blob.cx() + 10, blob.cy() - 10, color_strings[color_idx], color=color_new) # Подпись цвета

    # Отображение обработанного изображения на дисплее
    lcd.display(img)
    # Вывод текущей частоты кадров в консоль
    print(clock.fps())

```

13. Обнаружение человеческого тела

13.1 Цели эксперимента

Данный урок посвящён изучению функции обнаружения человеческого тела на платформе K210.

Путь к эталонному коду для эксперимента:

CanMV\05-AI\body_detect.py

13.2 Подготовка к эксперименту

Убедитесь, что модельные файлы успешно загружены на SD-карту по следующему пути:
/sd/KPU/head_body_detect/person_detect_v1.kmodel

13.3 Ход эксперимента

Заводская прошивка модуля уже включает модуль алгоритмов машинного зрения. Если ранее была установлена другая прошивка, перед проведением эксперимента необходимо восстановить заводскую версию.

Загрузка модели YOLO и инициализация детектора:

```
# Загрузка модели YOLO
body_kpu = KPU()
body_kpu.load_kmodel("/sd/KPU/head_body_detect/person_detect_v1.kmodel")
body_kpu.init_yolo2(anchor, anchor_num=len(anchor) // 2, img_w=320, img_h=240,
                     net_w=320, net_h=256, layer_w=10, layer_h=8,
                     threshold=0.5, nms_value=0.2, classes=len(names))
```

Основной цикл обработки кадров:

```
while True:
    # Ручная сборка мусора для предотвращения утечек памяти
    gc.collect()
    clock.tick()

    # Захват изображения и преобразование в формат, пригодный для YOLO
    img = sensor.snapshot()
    a = od_img.draw_image(img, 0, 0)
    od_img.pix_to_ai()

    # Выполнение расчётов с использованием нейросети
    body_kpu.run_with_output(input=od_img, getlist=False, get_feature=False)
    body_boxes = body_kpu.regionlayer_yolo2()

    if len(body_boxes) > 0:
        for l in body_boxes:
            # Отображение обнаруженных объектов
            a = img.draw_rectangle(l[0], l[1], l[2], l[3], color=(255, 0, 0))

    fps = clock.fps()
    a = img.draw_string(0, 0, "%2.1ffps" % (fps), color=(0, 60, 128), scale=2.0)
    lcd.display(img)
```

13.4 Результаты эксперимента

Подключите модуль K210 к компьютеру с помощью кабеля microUSB, нажмите кнопку подключения в среде CanMV IDE. После успешного соединения нажмите кнопку запуска для выполнения примера кода. Также можно сохранить код как main.ru и загрузить его на модуль K210 для автономной работы.

После завершения инициализации система отображает изображение с камеры на ЖК-дисплее. Обнаруженные люди на экране автоматически выделяются красными прямоугольниками.



13.5 Итоги эксперимента

В ходе эксперимента мы освоили реализацию базовой функции обнаружения человека на платформе K210 и получили практическое понимание работы модели YOLO в реальных приложениях. В дальнейших экспериментах можно экспериментировать с настройкой параметров модели (порог срабатывания threshold, значение подавления немаксимумов nms_value, размеры сетки) для оптимизации точности распознавания и скорости обработки изображений.

```

import sensor, image, time, lcd
from maix import KPU # KPU — Neural Processing Unit (специальный блок для ускорения нейросетей на K210)
import gc # Модуль сборки мусора для управления памятью

# =====
# 1. ИНИЦИАЛИЗАЦИЯ ОБОРУДОВАНИЯ
# =====

# Настройка дисплея и камеры — подготовка «глаз» системы
lcd.init()      # Включаем ЖК-дисплей для отображения результатов
sensor.reset()   # Перезагружаем сенсор камеры
sensor.set_pixformat(sensor.RGB565) # Устанавливаем формат цвета: 16-битный RGB (5 бит красный, 6 зелёный, 5 синий)
sensor.set_framesize(sensor.QVGA)    # Разрешение кадра: 320x240 пикселей (оптимальный баланс скорости и качества)
sensor.skip_frames(time = 1000)     # Пропускаем первые кадры, чтобы камера «адаптировалась» к освещению

# Создаём счётчик для измерения производительности (кадров в секунду)
clock = time.clock()

# Создаём буфер изображения для нейросети (размер 320x256 пикселей)
# Важно: нейросеть принимает изображения фиксированного размера, отличного от исходного кадра камеры
od_img = image.Image(size=(320,256))

# =====
# 2. НАСТРОЙКА НЕЙРОСЕТИ YOLO
# =====

# YOLO (You Only Look Once) — популярный алгоритм детекции объектов в реальном времени.
# Вместо поиска объектов по частям, он анализирует всё изображение за один проход.

# Список классов объектов, которые умеет распознавать модель (здесь только один класс — «тело»)
names = ['body']

# Anchor boxes (якорные рамки) — предопределённые шаблоны размеров и пропорций объектов
# Нейросеть «подгоняет» эти шаблоны под реальные объекты на изображении.
# Чем ближе якорь к форме объекта, тем точнее детекция.
# Формат: (ширина1, высота1, ширина2, высота2, ...)
anchor = (0.0978, 0.1758, 0.1842, 0.3834, 0.3532, 0.5982, 0.4855, 1.1146,
          0.8869, 1.6407, 1.2388, 3.4157, 2.0942, 2.1114, 2.7138, 5.0008,
          6.0293, 6.4540)

# Создаём объект для работы с нейропроцессором KPU
body_kpu = KPU()

# Загружаем предобученную модель с SD-карты
# Файл .kmodel — скомпилированная нейросеть, оптимизированная для K210
body_kpu.load_kmodel("/sd/KPU/head_body_detect/person_detect_v1.kmodel")

# Инициализируем детектор YOLO2 с указанными параметрами:
body_kpu.init_yolo2(
    anchor,           # Якорные рамки
    anchor_num=len(anchor)//2, # Количество якорей (каждый якорь = 2 числа: ширина и высота)
    img_w=320, img_h=240,   # Исходный размер кадра с камеры
    net_w=320, net_h=256,   # Размер входа нейросети (может отличаться от камеры!)
    layer_w=10, layer_h=8,   # Размер выходной «сетки» нейросети (10x8 ячеек для поиска объектов)
    threshold=0.5,         # Порог уверенности: объект считается найденным, если уверенность > 50%
    nms_value=0.2,         # NMS (Non-Maximum Suppression): подавление дублирующих рамок (чем меньше, тем строже
    фильтрация)          # Количество классов для распознавания
)

```

```

# -----
# ШАГ 3: Подготовка изображения для нейросети
# -----
# Копируем кадр в буфер нужного размера (320x256)
a = od_img.draw_image(img, 0, 0)
# Конвертируем пиксели в формат, понятный нейросети (AI-формат)
od_img.pix_to_ai()

# -----
# ШАГ 4: Запуск нейросети
# -----
# Передаём изображение в нейросеть для анализа
body_kpu.run_with_output(
    input=od_img,
    getlist=False,    # Не возвращать промежуточные данные
    get_feature=False # Не извлекать признаки (только детекция)
)

# Получаем результаты: список прямоугольников с координатами обнаруженных тел
# Формат каждого элемента: [x, у, ширина, высота, уверенность, класс]
body_boxes = body_kpu.regionlayer_yolo2()

# -----
# ШАГ 5: Визуализация результатов
# -----
if len(body_boxes) > 0: # Если найдены люди
    for l in body_boxes:
        # Рисуем красный прямоугольник вокруг каждого обнаруженного тела
        # Координаты: l[0]=x, l[1]=y, l[2]=ширина, l[3]=высота
        a = img.draw_rectangle(l[0], l[1], l[2], l[3], color=(255, 0, 0), thickness=3)

    # Отображаем текущую частоту кадров в левом верхнем углу
    # Это помогает оценить производительность системы
    fps = clock.fps()
    a = img.draw_string(0, 0, "%2.1ffps" % (fps), color=(0, 60, 128), scale=2.0)

    # Отображаем обработанное изображение на дисплее
    lcd.display(img)

except Exception as e:
    # Обработка ошибок: корректное освобождение ресурсов при сбое
    print("Ошибка:", e)
    body_kpu.deinit() # Освобождаем ресурсы нейропроцессора
    del body_kpu      # Удаляем объект
    gc.collect()       # Очищаем память

```

Полезные пояснения для новичков:

Понятие

Простое объяснение

YOLO

Нейросеть «видит» всё изображение сразу и за один проход находит все объекты.

Быстро, но требует хорошей тренировки.

Anchor boxes

Как «лекала» разного размера, которые нейросеть накладывает на изображение, чтобы найти объекты. Подбираются эмпирически под типичные размеры людей/голов.

Threshold (0.5)

Минимальный «уровень уверенности». Если нейросеть на 40% уверена, что это человек — рамку не нарисует.

NMS (0.2)

Если один человек обведён 3 рамками — NMS оставит только самую точную.

Значение 0.2 означает: удалять рамки, перекрывающиеся более чем на 80%.

Сборка мусора (gc.collect)

В микроконтроллерах память ограничена. Без регулярной очистки система «задохнётся» через несколько минут работы.

`pix_to_ai()`

Специальная конвертация пикселей в формат, оптимизированный для аппаратного ускорителя нейросетей на K210.

Советы для экспериментов:

Увеличьте `threshold` до 0.7 — детекция станет точнее, но могут пропадать дальние/маленькие объекты.

Уменьшите `nms_value` до 0.1 — уберёт «двойные» рамки, но может случайно удалить два близко стоящих человека.

Попробуйте другие модели — например, `head_detect` для поиска только голов (полезно для подсчёта людей).

Следите за FPS — если падает ниже 10, уменьшите разрешение камеры (`sensor.set_framesize(sensor.QQVGA) → 160x120`).

14. Обнаружение руки с помощью YOLO (YOLO Hand Detection)

14.1 Цели эксперимента

В данном уроке рассматривается функция обнаружения руки на платформе K210 с использованием модели YOLO.

Путь к эталонному коду для данного эксперимента:

CanMV\05-AI\yolo_hand_detect.py

14.2 Подготовка к эксперименту

Убедитесь, что файл модели успешно загружен на SD-карту по следующему пути:
/sd/KPU/yolo_hand_detect/hand_detect.kmodel

Важно: Для работы детектора обязательно требуется SD-карта с предварительно загруженной моделью. Без неё при запуске программы возникнет ошибка чтения файла.

14.3 Ход эксперимента

Заводская прошивка модуля уже включает модуль алгоритмов машинного зрения. Если ранее была установлена другая прошивка, перед проведением эксперимента необходимо восстановить заводскую версию.

```
lcd.init()
sensor.reset()
sensor.set_framesize(sensor.QVGA)      # Установка разрешения кадра: 320×240
sensor.set_pixformat(sensor.RGB565)    # Установка цветового формата: 16-битный RGB

# При необходимости можно включить вертикальное отражение изображения
# sensor.set_vflip(True)

# Создание буфера изображения для нейросети (размер 320×256 пикселей)
resize_img = image.Image(size=(320, 256))

# Якорные рамки (предобученные для распознавания рук)
anchor = (0.8125, 0.4556, 1.1328, 1.2667, 1.8594, 1.4889, 1.4844, 2.2000, 2.6484, 2.9333)

# Список распознаваемых классов (здесь только один — «рука»)
names = ['hand']

# Создание объекта для работы с нейропроцессором KPU (Kendryte Neural Network Processor)
hand_detector = KPU()

# Загрузка предобученной модели детектирования рук с SD-карты
hand_detector.load_kmodel("/sd/KPU/yolo_hand_detect/hand_detect.kmodel")

# Инициализация детектора YOLO2:
# img_w, img_h — размер исходного кадра с камеры (320×240)
# net_w, net_h — размер входа нейросети при обучении (320×256)
hand_detector.init_yolo2(
    anchor,
    anchor_num=len(anchor) // 2,
    img_w=320,
    img_h=240,
    net_w=320,
    net_h=256,
    layer_w=10,
    layer_h=8,
    threshold=0.68,   # Порог уверенности (68%)
    nms_value=0.3,   # Значение подавления немаксимумов
    classes=len(names)
)

while True:
    gc.collect() # Ручная сборка мусора для предотвращения утечек памяти
    img = sensor.snapshot() # Захват кадра с камеры

    # Масштабирование изображения и конвертация в формат для нейросети
    resize_img.draw_image(img, 0, 0,pix_to_ai)

    # Выполнение расчётов на нейропроцессоре KPU
    hand_detector.run_with_output(resize_img)

    # Получение результатов детектирования в формате YOLO2
    hands = hand_detector.regionlayer_yolo2()

    # Визуализация обнаруженных рук
    for hand in hands:
        # Рисование зелёного прямоугольника вокруг руки
        img.draw_rectangle(hand[0], hand[1], hand[2], hand[3], color=(0, 255, 0))
        # Отображение уровня уверенности (с точностью до 2 знаков)
        img.draw_string(hand[0] + 2, hand[1] + 2, "%6.2f" % (hand[5]), color=(0, 255, 0))
        # Подпись класса объекта
        img.draw_string(hand[0] + 2, hand[1] + 10, names[hand[4]], color=(0, 255, 0))

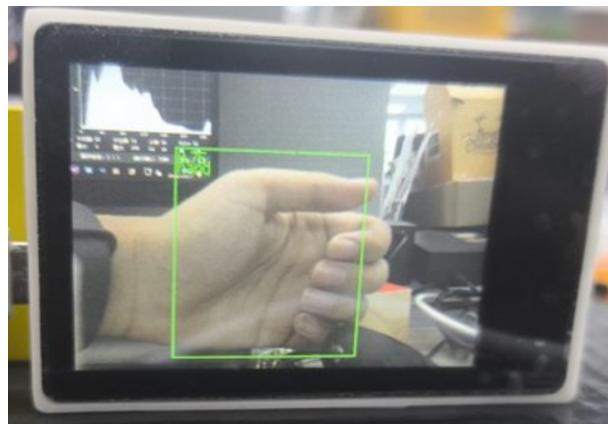
    # Отображение обработанного изображения на дисплее
    lcd.display(img)

# Примечание: следующие строки никогда не будут выполнены из-за бесконечного цикла while.
# В рабочем коде их следует разместить в блоке finally или обработать прерывание:
hand_detector.deinit()
del hand_detector
del img
gc.collect()
```

14.4 Результаты эксперимента

Подключите модуль K210 к компьютеру с помощью кабеля microUSB, нажмите кнопку подключения в среде CanMV IDE. После успешного соединения нажмите кнопку запуска для выполнения примера кода. Также можно сохранить код как main.ru и загрузить его на модуль K210 для автономной работы.

После завершения инициализации система отображает изображение с камеры на ЖК-дисплее. Обнаруженные кисти рук выделяются зелёными прямоугольниками с указанием уровня уверенности и подпись «hand».



Модель способна распознавать различные позы и формы кистей рук: открытую ладонь, сжатый кулак и другие жесты.

14.5 Итоги эксперимента

Для работы функции распознавания рук требуется SD-карта с загруженным файлом модели. Файл необходимо предварительно скопировать на карту памяти, после чего вставить её в слот модуля K210. При отсутствии файла или ошибке в пути к нему будет выведено сообщение об ошибке.

В ходе эксперимента мы освоили реализацию функции детектирования рук на платформе K210 с использованием архитектуры YOLO. Следует учитывать, что из-за компактного размера модели (оптимизированной для встраиваемых систем с ограниченными ресурсами) точность распознавания может быть снижена в сложных условиях освещения или при частичном перекрытии руки другими объектами.

Для повышения надёжности рекомендуется:

Обеспечить равномерное освещение рабочей зоны

Поддерживать расстояние до руки в пределах 30–80 см

При необходимости увеличить порог threshold для снижения ложных срабатываний

15. Распознавание автомобильных номерных знаков (License Plate Recognition)

В данном эксперименте рассматривается функция распознавания автомобильных номеров на платформе K210.

Путь к эталонному коду для данного эксперимента:

CanMV\05-AI\car_licenseplate_recog_cn.py

15.1 Цели эксперимента

Основная цель данного эксперимента — изучение функции распознавания номерных знаков автомобилей на модуле K210.

15.2 Подготовка перед экспериментом

ВАЖНО! Модельные и весовые файлы, используемые в этом примере, имеют большой размер. Перед запуском обязательно прошайте облегчённую версию прошивки (lite version firmware), иначе система зависнет.

```
/sd/KPU/car_licenseplate_recog/lp_detect.kmodel    # Модель детекции номера (YOLO)
/sd/KPU/car_licenseplate_recog/lp_recog.kmodel    # Модель распознавания символов
/sd/0xA00000.Dzk                                # Шрифтовая библиотека для китайских иероглифов
/sd/KPU/car_licenseplate_recog/lp_weight.bin     # Весовые коэффициенты для постобработки
```

Примечание:

- .kmodel — скомпилированная нейросеть для K210
- .Dzk — шрифтовой файл в кодировке UTF-8 (поддержка китайских иероглифов)
- .bin — бинарные веса для преобразования выхода нейросети в читаемые символы

15.3 Ход эксперимента

Заводская прошивка модуля уже содержит встроенный модуль алгоритмов AI-зрения. Если ранее была установлена другая прошивка, перед началом эксперимента необходимо прошить модуль обратно заводской (lite-версией).

Инициализация библиотеки шрифтов и параметров модели

```
image.font_load(image.UTF8, 16, 16, "/sd/0xA00000.Dzk")
# image.font_load(image.UTF8, 16, 16, 0xA00000)
```

```

# Загрузка шрифтовой библиотеки для отображения китайских иероглифов
# Параметры: кодировка UTF8, ширина=16px, высота=16px, путь к файлу шрифта
image.font_load(image.UTF8, 16, 16, "/sd/0xA00000.Dzk")
# Альтернативный вариант (если шрифт прошит в память):
# image.font_load(image.UTF8, 16, 16, 0xA00000)

# Список китайских провинций (первый символ номера)
province = ("")

# Допустимые символы для остальных позиций номера: буквы (без I и O) + цифры
ads = ('A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'J', 'K', 'L', 'M', 'N', 'P',
       'Q', 'R', 'S', 'T', 'U', 'V', 'W', 'X', 'Y', 'Z',
       '0', '1', '2', '3', '4', '5', '6', '7', '8', '9')

# Список классов для детектора (пустой — модель обучена только на номерах)
names = []

# Anchor boxes для детекции номеров (оптимизированы под прямоугольную форму номера)
anchor = (8.30891522166988, 2.75630994889035, 5.18609903718768, 1.7863757404970702,
          6.91480529053198, 3.825771881004435, 10.218567655549439, 3.69476690620971,
          6.4088204258368195, 2.38813526350986)

# === ЭТАП 1: ДЕТЕКЦИЯ НОМЕРА (используем YOLO) ===
lp_det_kpu = KPU() # KPU = Kendryte Processing Unit (нейроускоритель)
lp_det_kpu.load_kmodel("/sd/KPU/car_licenseplate_recog/lp_detect.kmodel")
lp_det_kpu.init_yolo2(
    anchor,
    anchor_num=len(anchor) // 2, # 5 якорных рамок
    img_w=320, img_h=240, # Размер кадра с камеры
    net_w=320, net_h=240, # Размер входа нейросети
    layer_w=20, layer_h=15, # Сетка детекции: 20×15 = 300 ячеек
    threshold=0.7, # Высокий порог для снижения ложных срабатываний
    nms_value=0.3, # Подавление дублирующих рамок
    classes=len(names)
)
# === ЭТАП 2: РАСПОЗНАВАНИЕ СИМВОЛОВ ===
lp_recog_kpu = KPU()
lp_recog_kpu.load_kmodel("/sd/KPU/car_licenseplate_recog/lp_recog.kmodel")
lp_recog_kpu.lp_recog_load_weight_data("/sd/KPU/car_licenseplate_recog/lp_weight.bin")

```

Вспомогательная функция для расширения рамки вокруг номера:

```
def extend_box(x, y, w, h, scale):
```

```
    """
```

Расширяем рамку вокруг номера для лучшего распознавания символов.

Номера часто имеют узкие границы — расширение захватывает полный контекст.

Параметры:

x, y — координаты верхнего левого угла

w, h — ширина и высота исходной рамки

scale — коэффициент расширения (0.08 = +8% с каждой стороны)

Возвращает:

x1, y1, cut_img_w, cut_img_h — координаты и размеры расширенной области

```
    """
```

```
x1_t = x - scale*w
```

```
x2_t = x + w + scale*w
```

```
y1_t = y - scale*h
```

```
y2_t = y + h + scale*h
```

```
x1 = int(x1_t) if x1_t > 1 else 1
```

```
x2 = int(x2_t) if x2_t < 320 else 319
```

```
y1 = int(y1_t) if y1_t > 1 else 1
```

```
y2 = int(y2_t) if y2_t < 256 else 255
```

```
cut_img_w = x2 - x1 + 1
```

```
cut_img_h = y2 - y1 + 1
```

```
return x1, y1, cut_img_w, cut_img_h
```

Основная программа: детекция номера и распознавание текста



京A·9Z568



沪C·73D56



苏E·A987K



豫C·6T52N



豫C·6T52N

```

try:
    while True:
        gc.collect()          # Очистка памяти (критично для K210!)
        clock.tick()         # Запуск таймера для расчёта FPS
        img = sensor.snapshot() # Захват кадра с камеры

        # === ШАГ 1: ДЕТЕКЦИЯ РАСПОЛОЖЕНИЯ НОМЕРА ===
        lp_det_kpu.run_with_output(img)
        lps = lp_det_kpu.regionlayer_yolo2() # Получаем список рамок с номерами

        for lp in lps:
            # Расширяем рамку вокруг номера (+8% с каждой стороны)
            x, y, w, h = extend_box(lp[0], lp[1], lp[2], lp[3], 0.08)
            img.draw_rectangle(x, y, w, h, color=(0, 255, 0)) # Зелёная рамка

        # === ШАГ 2: РАСПОЗНАВАНИЕ СИМВОЛОВ ===
        lp_img = img.cut(x, y, w, h)      # Вырезаем область номера
        resize_img = lp_img.resize(208, 64) # Масштабируем до размера входа модели (208x64)
        resize_img.replace(hmirror=True)   # Горизонтальное отражение (для корректной ориентации)
        resize_img.pix_to_ai()           # Конвертация в формат для KPU

        lp_recog_kpu.run_with_output(resize_img)
        output = lp_recog_kpu.lp_recog() # Получаем "сырые" выходы нейросети

        # Преобразуем выходы в индексы символов (выбираем максимальную активацию для каждой позиции)
        lp_indices = [o.index(max(o)) for o in output]

        # Формируем читаемую строку номера:
        # Формат китайского номера: [Провинция] [Буква] - [5 символов: буквы/цифры]
        show_lp_str = "%s %s-%s%s%s%s%s" %
            province[lp_indices[0]], # Первый символ — провинция (иероглиф)
            ads[lp_indices[1]],     # Второй символ — буква региона
            ads[lp_indices[2]], ads[lp_indices[3]], ads[lp_indices[4]],
            ads[lp_indices[5]], ads[lp_indices[6]] # Остальные 5 символов
        )

        # ВАЖНО: перезагружаем шрифт перед выводом китайских иероглифов!
        # Без этого шага иероглифы могут отображаться как "кракозябы"
        image.font_load(image.UTF8, 16, 16, "/sd/0xA00000.Dzk")

        # Отображаем распознанный номер НАД рамкой (с отступом 20 пикселей)
        img.draw_string(
            x + 2, y - 20 - 2,
            "%s" % show_lp_str,
            color=(255, 0, 0), # Красный цвет текста
            x_spacing=2,       # Расстояние между символами
            mono_space=False, # Пропорциональный шрифт
            scale=2           # Масштаб текста
        )

        # Освобождаем память (важно для предотвращения утечек)
        del lp_img
        del resize_img
        gc.collect()

    lcd.display(img) # Отображение результата на дисплее

except Exception as e:
    print("Ошибка:", e)
    lp_det_kpu.deinit()
    lp_recog_kpu.deinit()
    del lp_det_kpu
    del lp_recog_kpu
    gc.collect()

```

15.4 Результаты эксперимента

Подключите модуль K210 к компьютеру с помощью кабеля microUSB, нажмите кнопку подключения в среде CanMV IDE. После успешного соединения нажмите кнопку запуска для выполнения примера кода. Также можно сохранить код как main.py и загрузить его на модуль K210 для автономной работы.

После завершения инициализации система отображает изображение с камеры на ЖК-дисплее. Обнаруженные автомобильные номера выделяются зелёными прямоугольниками, а над рамкой отображается распознанный текст номера в формате китайских автомобильных регистрационных знаков.



15.5 Итоги эксперимента

Для работы функции распознавания номеров требуется SD-карта с предварительно загруженными файлами моделей, шрифтовой библиотеки и весовыми коэффициентами. Файлы необходимо разместить по указанным путям до запуска программы. При отсутствии файлов или ошибке в пути будет выведено сообщение об ошибке. Модельные и весовые файлы, используемые в данном примере, имеют значительный размер. Поэтому перед запуском необходимо прошить облегчённую версию прошивки (lite version firmware), иначе система исчерпает доступную память и зависнет. В ходе эксперимента мы освоили двухэтапный подход к распознаванию номеров на платформе K210:

Детекция расположения номера с помощью YOLO

Распознавание символов с помощью специализированной CNN-модели

Ограничения текущей реализации:

Модель обучена только на китайские номера (иероглиф провинции + латинские буквы/цифры)

Точность распознавания зависит от:

Угол наклона номера (оптимально — фронтальное расположение)

Освещения (избегайте бликов на номерной табличке)

Расстояния (рекомендуется 50–150 см от камеры)

Из-за компактного размера модели (оптимизированной для встраиваемых систем) возможны ошибки при плохом качестве изображения

Для повышения точности можно:

Обучить собственную модель на наборе данных с номерами нужного формата (российские, европейские и т.д.)

Добавить предварительную обработку изображения (коррекция перспективы, повышение контраста)

Использовать ансамбль моделей для критически важных приложений

16. Самостоятельное обучение модели и её использование

16.1. Цели эксперимента

В этом уроке рассматривается, как самостоятельно обучить модель и запустить её на K210.

16.2.1. Обучение модели

Перед началом обучения убедитесь, что на модуле установлена заводская прошивка Yahboom, иначе обученная модель не сможет корректно работать.

Перейдите на сайт обучения моделей:

<https://www.kendryte.com/en/products>

Зарегистрируйте собственный аккаунт.

Ссылка на стартовую страницу обучения моделей:

<https://www.kendryte.com/en/training/start>

Важные моменты про платформу

Платформа бесплатна для экспериментов, но возможны ограничения по ресурсам (одновременные задачи и количество изображений).

После обучения набор файлов с моделью (*.kmodel) будет отправлен на вашу почту и доступен для скачивания.

Поддерживаются несколько типов задач (например, детекция и классификация).

Обучение выполняется в облаке, так что на вашей машине ничего не нужно устанавливать.

Альтернативы, если не удаётся зайти

Если облачная тренировка на сайте недоступна или неудобна:

AI Cube — локальный инструмент от Canaan для создания/обучения моделей на вашем компьютере (Windows или Ubuntu). Это даст больше контроля над процессом.

nncase — CLI-инструмент для преобразования ваших TFLite/ONNX моделей в .kmodel, чтобы запускать их на K210 без облака.

Ссылка на Dataset <https://www.kendryte.com/en/resource/training,k210>

Нажмите Dataset → Create Dataset, введите имя датасета, затем выберите тип разметки Image Detection и нажмите Submit.

Модель, обучаемая в этом разделе, предназначена для распознавания трёх типов транспорта:

[самолёты, автомобили и корабли].

Выберите только что созданный датасет.

Нажмите Upload Image, загрузите изображения и задайте соответствующие метки.

Нажмите кнопку Label, чтобы перейти в интерфейс разметки изображений, и разметьте объекты соответствующими метками.

С помощью мыши выделите объект рамкой и нажмите Save.

Найдите созданный датасет в разделе Dataset и нажмите кнопку Training.

На странице параметров обучения можно использовать значения по умолчанию или настроить их вручную.

Дождитесь завершения обучения.

Примечание:

Иногда прогресс обучения может зависнуть на 0% — пожалуйста, подождите.

После завершения обучения скачайте модель.

Файл *.kmodel — это обученная модель,

det.py — пример кода для использования данной модели.

Скопируйте файл kmodel на SD-карту.

В этом примере модель размещается по пути:

/sd/KPU/train_model_and_use/det.kmodel

Примечание:

Если итоговое качество модели неудовлетворительное, рекомендуется предварительно привести все изображения обучающего набора к одинаковому размеру.

16.2.2. Использование (вызов) модели

Откройте файл det.py из архива, загруженного на предыдущем этапе, в IDE.

Измените путь загрузки модели:

```
kpu.load_kmodel('/sd/KPU/train_model_and_use/det.kmodel')
```

Измените содержимое массива labels в соответствии с метками вашей модели (см. файл label.txt):

```
# Названия классов, указываются в порядке из label.txt
labels = ["plane", "ship", "car"]
```

Задайте anchors, используя вторую строку из файла anchor.txt:

```
anchor = (9.50, 4.00, 7.97, 5.06, 9.69, 4.81, 7.59, 6.58, 9.36, 6.02)
```

Инициализация KPU

```
kpu = KPU()
# Загрузка модели с SD-карты или из flash
kpu.load_kmodel('/sd/KPU/train_model_and_use/det.kmodel')
# kpu.load_kmodel(0x300000, 584744)

kpu.init_yolo2(
    anchor,
    anchor_num=int(len(anchor)/2),
    img_w=320,
    img_h=240,
    net_w=320,
    net_h=240,
    layer_w=10,
    layer_h=8,
    threshold=0.1,
    nms_value=0.3,
    classes=len(labels)
)
```

Основной цикл обнаружения объектов

```
while(True):
    gc.collect()
    clock.tick()
    img = sensor.snapshot()
    kpu.run_with_output(img)
    dect = kpu.regionlayer_yolo2()
    fps = clock.fps()

    if len(dect) > 0:
        for l in dect:
            img.draw_rectangle(l[0], l[1], l[2], l[3], color=(0,255,0))
            info = "%s %.3f" % (labels[l[4]], l[5])
            img.draw_string(l[0], l[1], info, color=(255,0,0), scale=2.0)
            print(info)

    img.draw_string(0, 0, "%2.1ffps" % fps, color=(0,60,255), scale=2.0)
    lcd.display(img)
```

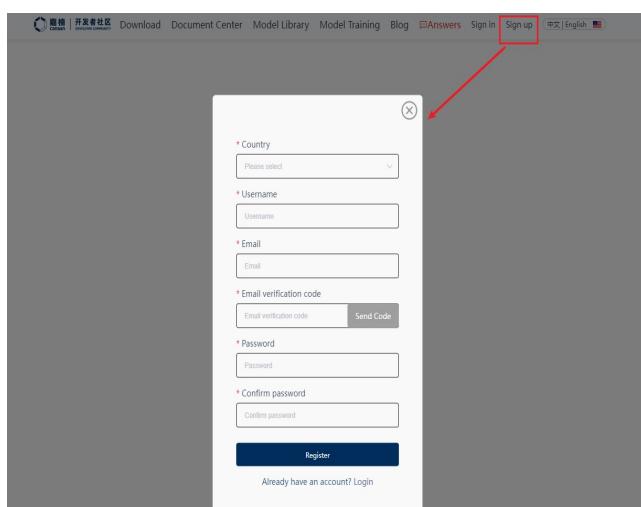
16.3. Результаты эксперимента

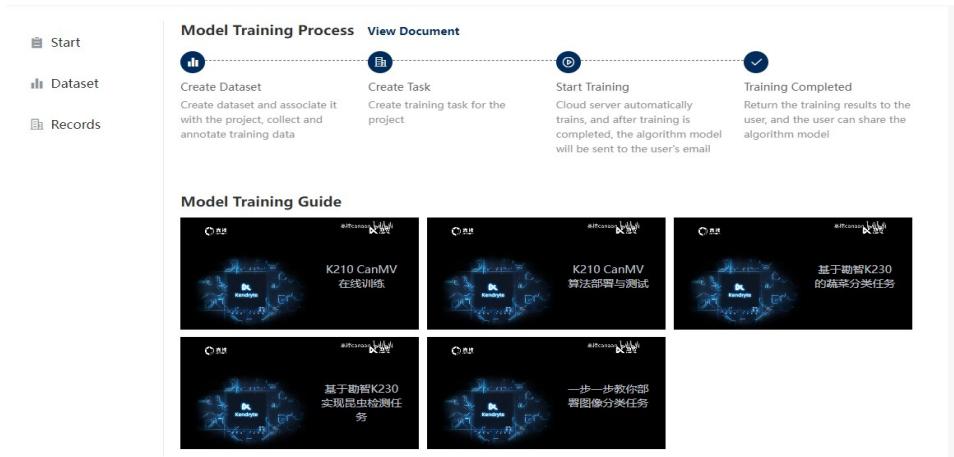
Подключите модуль K210 к компьютеру с помощью кабеля microUSB, нажмите кнопку Connect в CanMV IDE, затем нажмите Run для запуска примера.

Также можно сохранить код как main.py на модуле K210 и запускать его напрямую.

После инициализации системы LCD-дисплей выводит изображение с камеры и отмечает на нём объекты ship, plane и car.

Пример результата показан на изображениях ниже.





Dataset

Create Dataset

Dataset Name: car_plane_ship_detect

Annotation Type: Image Detection

Note: K210 only supports Image Classification and Image Detection

Operation

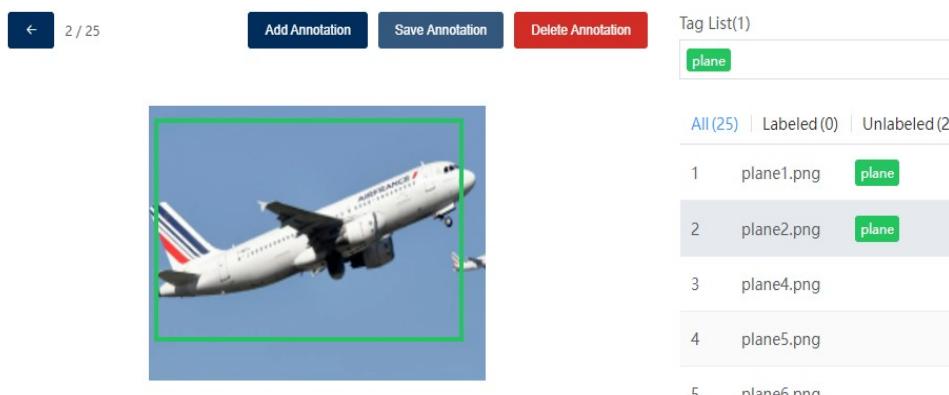
ID	Time	Operation
2895	2024-10-23 15:31:2	Train Edit Delete
1940	2024-10-23 17:36:0	Train Edit Delete
1880	2024-10-23 08:55:4	Train Edit Delete
1840	2024-10-23 11:22:3	Train Edit Delete

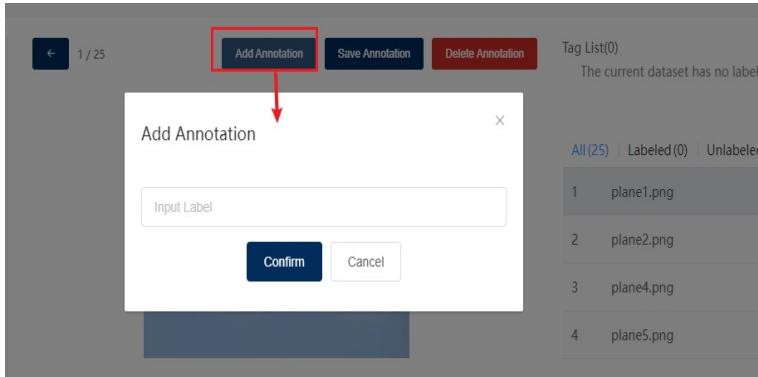
Submit

ID	Name	Annotation Type	Create Time	Update Time	Operation
9971	car_plane_ship_detect	Image Detection	2024-10-23 15:31:3 5	2024-10-23 15:31:3 5	Train Edit Delete

car_plane_ship_detect

[Image Detection](#) [Total Images 0](#) [Upload Images](#) [Upload Archive](#) [Archive Format Description](#) [Annotate](#) [Train](#)





Create Task

Task Name: train_car_plane_ship_detect

Platform: k210 k230

Iterations: 240

Batch Size: 8 16 24 32

Learning Rate: 0.001

Label Box Limit: 5

Confirm

Training Name: car_plane_ship_detect

Training ID: 8861

Training Status: **Training Completed**

100%

Training Parameters

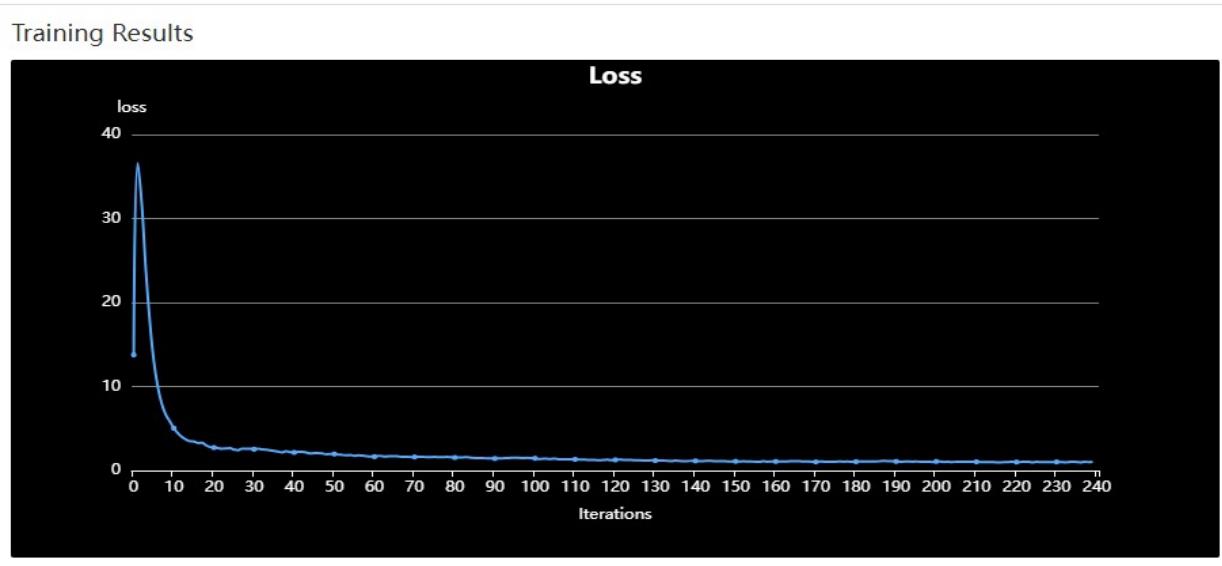
Epochs	240
Batch Size	32
Learning Rate	0.001
Label Box Limit	5

Training Logs

```

[ 0.00s ] 100%
3.23s
4. 5. Quantize graph...
5. Lowering...
6. Optimize Pass 3...
7. Generate code...
Plan buffers...
Emit code...
Working memory usage: 476160 B

SUMMARY
INPUTS
0      inputs  1x3x240x320
OUTPUTS
0      MobilenetV1/detect_layer/yolo_out/Conv/BiasAdd
1x40x8x10
  
```



Start
Dataset
Records

Training Record

- Tip: 1. After the training is completed, the model will be automatically sent to the registered email.
2. Each user supports only one training task running at the same time.
3. The maximum training duration for a single task is 24 hours. If it exceeds the time, the task will be terminated early according to the server resource situation.
4. The training duration is strongly related to the number of files and training iterations of your dataset. If the training time is too long, consider reducing the corresponding quantity.

ID	Name	Dataset	Type	Chip	Status	Update Time	Operation
8861	car_plane_ship_detect	car_plane_ship_detect	Image Detection	k210	✓Completed	2024-10-23 ...	Detail Download Delete

文件夹				
det_results				
anchor.txt		1 KB	1 KB	文本文档
canmv-f7c35aa1775e2ee1e2fbffff634b06a30ee25c.zip		6.4 MB	6.4 MB	360压缩 ZIP 文件
det.kmodel		548.0 KB	548.0 KB	KMODEL 文件
det.py		1.8 KB	1.8 KB	Python 源文件
label.txt		1 KB	1 KB	文本文档
train_loss.txt		3.2 KB	3.2 KB	文本文档



6. Расширенные функции (продаются отдельно)