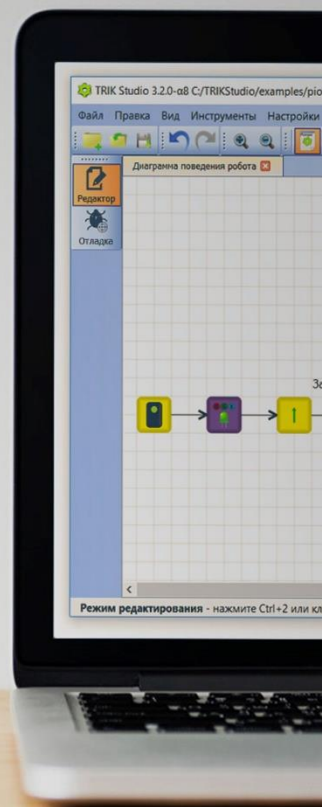




ФОНД СОДЕЙСТВИЯ РАЗВИТИЮ
ВОЕННОГО ОБРАЗОВАНИЯ



GEOSCAN

«Робототехника и управление беспилотными авиационными системами»

Автор-составитель:
Азибаев Р.С.



Методический материал является результатом сотрудничества компании «Геоскан» и Фонда содействия развития военного образования с целью внедрения новых технологий в образовательный процесс довузовских военных учреждений Министерства обороны.

Учебный курс «Робототехника и управление беспилотными авиационными системами» адресован как преподавателям, так и учащимся специализированных военно-учебных заведений для изучения по программе дополнительного военного образования.

В ходе курса изучаются такие разделы, как основы программирования и автономного полёта.

Квадрокоптер «Геоскан Пионер», на основе которого построен курс, признан лучшим учебным отечественным оборудованием и награждён знаком «ВЫБОР ПЕДАГОГОВ».

Оглавление

Введение.....	4
Занятие 1. OpenMV.....	6
Занятие 2. Разбор примеров OpenMV.....	10
Занятие 3. OpenMV, рисуем на экране.....	12
Занятие 4. Распознавание объектов.....	15
Занятие 5. Распознавание сложных объектов.....	17
Занятие 6. Подключение OpenMV к полетной плате Пионер, протоколы связи. Программирование камеры для полета по ArUco меткам.....	21
Занятие 7. Программирование квадрокоптера для полета по ArUco меткам. ...	29
Занятие 8. Контрольный рубеж.	36

Введение

Перед вами 3 модуль методического пособия от ГК «Геоскан» совместно с Фондом содействия развитию военного образования (ФСРВО).

Целью пособия является обучение основам машинного зрения, обобщенным понятиям по машинному обучению и нейросетям, а также истории возникновения всех этих понятий.

По традиции, большая часть затронутых примеров выложена в электронном виде.

Программный код, используемый в пособии, выделен курсивом, а над ним находится название файла.

../main.py

red_led = pyb.LED(1)

Перед началом работы необходимо скачать используемые IDE: OpenMV IDE, PioneerStation. Все ссылки есть в документации Пионера и на Github.

<https://docs.geoscan.aero>

https://github.com/Slond/pioneer_edu

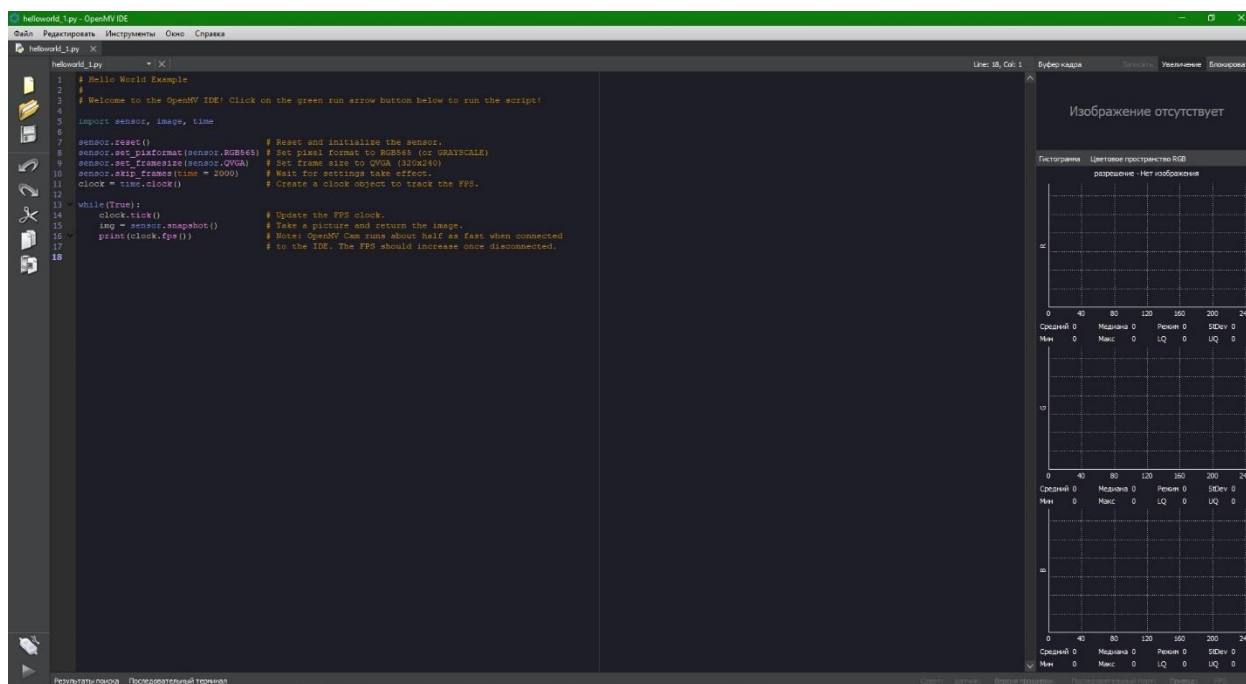
Данное пособие рассчитано на обучающихся старших классов довузовских учреждений военного образования, имеющих базовые понятия алгебры и планиметрии. Для выполнения занятий потребуются квадрокоптер Геоскан Пионер Базовый и дополнительные модули: камера машинного зрения OpenMV, модуль захвата груза, LED-модуль, а также модуль для используемой системы позиционирования (в большинстве случаев будет использоваться OPT).

Термины «машинное зрение» и «нейросеть» в наше время слышны отовсюду. Если ввести что-либо из этого в браузер, то вы узнаете, что нейросети, уже, кажется, научились делать все и делает это на порядок лучше, чем человек. Здесь и далее мы обсудим, почему это происходит, как работают нейросети, создадим собственную нейросеть, обучим ее на готовых данных, а также собственноручно приблизим восстание машин.

Занятие 1. OpenMV

Первым делом необходимо подготовить ПК к работе с камерой машинного зрения. Для этого необходимо скачать среду разработки, OpenMV IDE (на момент написания последняя версия 2.8.1).


Большая часть кода в этом пособии будет написана на новом для вас языке программирования – Python, поэтому сначала мы будем разбираться с языком на встроенных примерах OpenMV IDE.

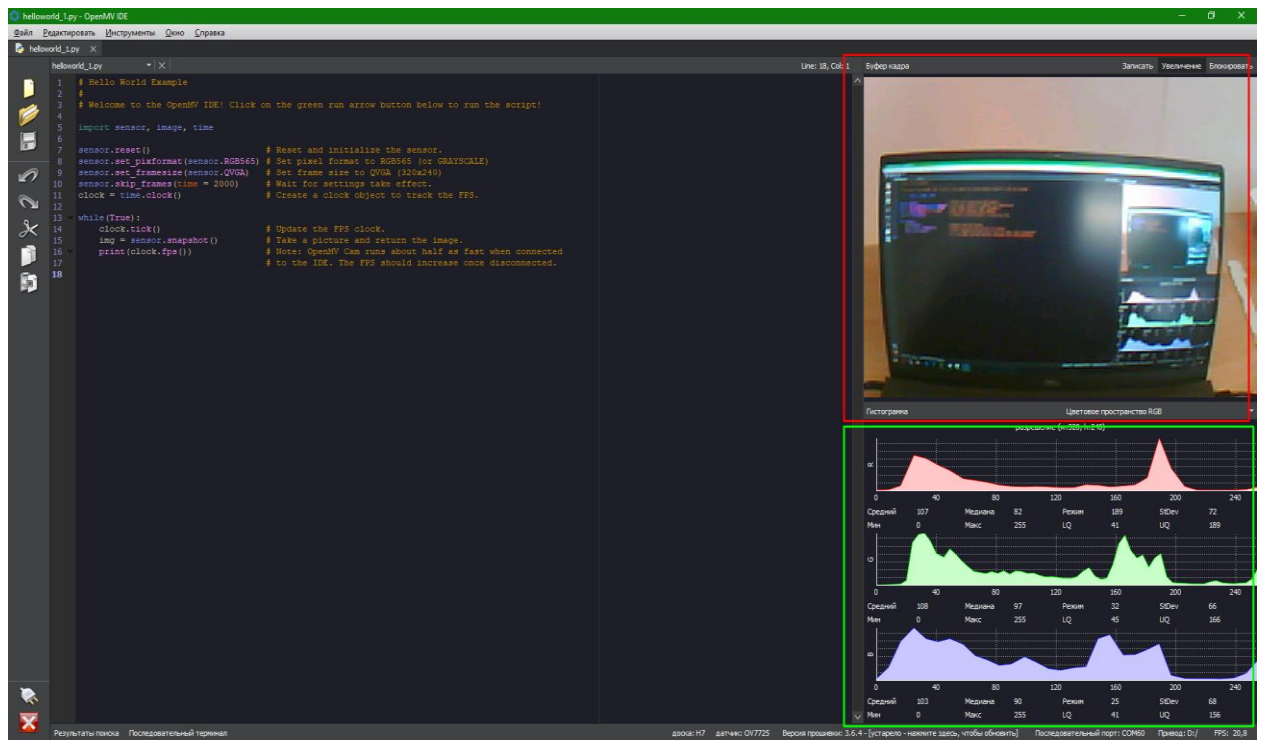


Вид главного меню программы

Для начала необходимо подключить камеру OpenMV через USB.



Снизу слева необходимо нажать на , это соединит программу с камерой. Далее нажимаем на зеленую кнопку запуска скрипта, расположенную слегка ниже.



После запуска появится два окна: в красной рамке изображение с камеры, а в зеленой гистограмма изображения.

Гистограмма изображения – график распределения пикселей изображения с разной яркостью. По горизонтали значение яркости каждого цвета (значения от 0 до 255), по вертикали количество пикселей данного цвета.

Обычно гистограмма нужна фотографам, так как она показывает яркость конкретного кадра, на экране, допустим, показан крайне плохой вариант фото, так как два пика находятся в области очень малого освещения (слева) и переосвещения (справа). Идеальное фото, где гистограмма выглядит слегка пологим колоколом Гаусса.

Теперь необходимо разобрать код.

Python – высокоуровневый динамически типизируемый язык программирования, сила которого заключается в простоте понимания и большом количестве подключаемых библиотек.

Каждая программа начинается с импорта всех необходимых библиотек, т.е. мы говорим программе, какие функции будем использовать. В данном коде это строка 5

(*import sensor, image, time*). Т.е. используются встроенные библиотеки для работы с камерой, изображением с камеры и временем.

Далее, строки 7-10 обновляют параметры камеры, а именно цветовую гамму и разрешение.

Строка 11 отвечает за получение данных о FPS (frame per second – число кадров в секунду).

Начиная с 13 строки запускается бесконечный цикл, используемый для постоянного вывода информации об изображении. (14 – обновить данные о FPS, 15 – обновить показываемую картинку, 16 – вывести FPS). Для вывода данных о FPS, необходимо нажать снизу «Последовательный терминал».

Чтобы остановить программу, нужно нажать на кнопку «Стоп» (нижний левый угол).

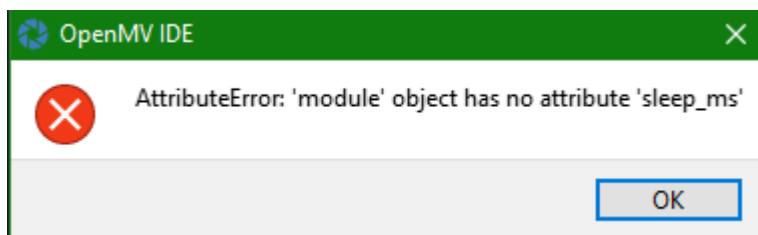
Далее рассмотрим один пример. Для этого необходимо сверху войти во вкладку «Файл», далее «примеры», «OpenMV», «Basics», «Main.py».

Данный пример учит использовать светодиод, находящийся на плате камеры.

В начале программы также прописаны все импорты. На строках 9-10 инициализируется светодиод и USB порт, используемый камерой.

В комментариях к коду прописан цвет светодиода, он указывается в скобках у метода *pyb.LED(#)*.

Однако, после запуска кода открывается ошибка.



Она говорит о том, что у библиотеки `time` нет метода `"sleep_ms"`. Зато есть метод `"sleep"`, в котором в скобках указывается время сна в мс.

Соответственно, необходимо каждый `"sleep_ms"`, заменить просто на `"sleep"`.

После этого светодиод на камере начнет мигать зеленым цветом.

Занятие 2. Разбор примеров OpenMV

Ученые достаточно давно пытались придумать системы компьютерного зрения, однако этому мешала маломощная техника, так как обработка видеопотока достаточно нетривиальная задача. Лишь в 1980-х началось исследование возможностей машин обрабатывать видео.

Это вылилось в 3 тесно связанные области: компьютерное зрение, машинное зрение и обработка изображений. Эти области настолько близки, что их редко получается разделить, однако:

- Компьютерное зрение – это обработка трехмерных моделей, созданных из одного или нескольких изображений
- Обработка изображений – работа с двумерными изображениями, в основном преобразование пикселей
- Машинное зрение – использование возможностей остальных областей в промышленности.

Компьютерное зрение активнее всего развивается в отраслях с большим количеством графического материала.

Основные области:

- Медицина. Нейросети, обученные на медицинских данных (различные рентгеновские снимки), распознают многие заболевания на порядок лучше и раньше, чем врачи (для рентгена легких процент точности врача ~80%, для машины ~97%)
- Военная промышленность. Системы распознавания «свой»/«чужой», системы наведения, разведка с помощью БПЛА, все большую роль в военных действиях играют именно автономные беспилотники с машинным зрением.
- Промышленность. Контроль производства, проверка качества продукции, измерение ориентации и положения продукции.

Основные задачи машинного зрения:

- Распознавание объекта
- Идентификация объекта
- Обнаружение
- Поиск изображения с заданным содержанием
- Оптическое распознавание знаков или символов
- Восстановление изображений

На данном занятии речь в-основном пойдет об обработке изображений.

Откроем «Файл», далее «примеры», «OpenMV», «Image Filter», «blur_filter.py».

Данный скрипт создает черно-белую картинку и слегка размывает ее.

Здесь только два отличия от первого скрипта. 8 строка – открывает фото в черно-белом цвете (для цветного изображения смените GRAYSCALE на RGB565). Далее отличие на строке 18, `img.gaussian(1)`. Наведите мышкой на слово "Gaussian", чтобы прочесть, что делает данная функция. Попробуйте самостоятельно изменить степень размытия.

Попробуйте запустить все примеры из данного блока примеров, чтобы понять, какое изображение можно получить с различными фильтрами.

Каждый фильтр имеет свой диапазон применения.

Занятие 3. OpenMV, рисуем на экране

Очень часто графической информации недостаточно для понимания всей картины происходящего. Python позволяет выводить дополнительно любой текст на видеопоток.

Для этого, запустим пример из папки «Drawing».

Хорошим примером для начала будет «line_drawing.py».

Данный пример рисует случайные прямые случайным цветом.

Первые 16 строк нам уже известны, импорты, инициализация, получение видеопотока.

Далее начинается счетный цикл, который создает 9 прямых. Прямую можно задать либо формулой $y = kx + b$, либо с помощью двух точек. Соответственно, строки 19-25 генерируют 4 случайных числа и 3 числа для цвета (красный, зеленый, синий).

Далее новый метод `draw_line`, в котором мы указываем координаты точек, через которые проходит отрезок, ее цвет и толщину.

После этого попробуем создать свой код, который выведет FPS на экран, а также подчеркнет эту надпись.

Для этого нам нужна новая команда `img.draw_string`

OpenMV IDE очень помогает в работе, предлагая все варианты команд, а также подсказывая, какие именно параметры нужны в скобках.

```

import sensor, image, time, pyb

sensor.reset()
sensor.set_pixformat(sensor.RGB565)
sensor.set_framesize(sensor.QVGA)
sensor.skip_frames(time = 2000)
clock = time.clock()

while(True):
    clock.tick()

    img = sensor.snapshot()

    # Character and string rotation can be done at 0, 90, 180, 270, and etc. degrees.
    img.draw_string(0, 200, str(clock.fps()), color = (255, 0, 0), scale = 2, mono_space = False,
                   char_rotation = 0, char_hmirror = False, char_vflip = False,
                   string_rotation = 0, string_hmirror = False, string_vflip = False)

    img.draw_line(0,220,100,220,(0,255,0),2)

    print(clock.fps())

```

Первые строки, все до `img = sensor.snapshot()` уже знакомы.

Далее прописываем `draw_string`, который имеет следующие параметры:

- `x,y` – координаты верхнего левого угла надписи
- `String` – строка, которая будет выведена (в формате строки, т.е. перед `clock.fps()` необходима команда `str`, которая переведет число из формата десятичной дроби в формат строки)
- Цвет надписи
- `Scale` – размер шрифта
- Об остальных параметрах можно прочесть, наведя курсор на функцию, в основном это функционал, позволяющий переворачивать строку

Далее уже знакомая команда `draw_line`, из параметров: координаты 2х точек, цвет и толщина.



Результат выполнения скрипта.

Далее попробуйте самостоятельно дорисовать прямоугольник вокруг числа FPS, либо линиями, либо используя команду `img.draw_rectangle`.

Занятие 4. Распознавание объектов

Прежде чем переходить к распознаванию объектов, сперва следует понять, как именно компьютер распознает очертания предметов в видеопотоке.

Краем (границей/очертанием) предмета называют такую кривую на изображении, вдоль которой происходит резкое изменение яркости, цвета или глубины сцены.

Однако часто возникает ситуация, что компьютеру не нужно обрабатывать все изображение. Это позволяет экономить ресурсы и обрабатывать необходимую информацию быстрее. Допустим, камеры на дорогах не обрабатывают все свое поле зрения, а только ту часть, которая смотрит на полосы. Это подталкивает к введению термина ROI – Region of interest (регион интересов – интересующая часть изображения). Чаще всего это некий прямоугольник на фотографии/видеопотоке.

Основной метод распознавания границ – детектор границ Кенни. Этот алгоритм был предложен Джоном Кенни в 1986 году, после многолетних математических вычислений. До сих пор это лучший вариант распознавания, принятый во всем мире стандартом.

Основные шаги детектора:

- Убрать шум и лишние детали из изображения (Выделить ROI, отфильтровать изображение)
- Рассчитать градиент изображения с помощью оператора Собеля (не волнуйтесь, в дискретную алгебру никто не полезет, компьютер рассчитает сам)
- Сделать края на изображении тоньше
- Связать отдельные края в контур

Если сильно упростить, то детектор Кенни создает градиент изображения, и сводит все к небольшим пиксельным квадратам. Далее, проверяет, есть ли перепады

по яркости, и, если есть, какой угол у границы. И после по заданным параметрам сшивает несколько краев в один контур.

Для работы с детектором Кенни откройте пример в папке «Feature detection», «Edges.py».

Суть примера находится на 17 строке `img.find_edges`. Внутри требуются два параметра – вид детекции (используем алгоритм Кенни) и параметр `threshold` (порог). Порог разделяется на нижний и верхний. Все значения градиента ниже нижнего порога не учитываются. Все значения градиента выше верхнего порога считаются краем. Все значения между нижним и верхним порогом учитываются только тогда, когда пиксель с данным градиентом касается пикселя с значением выше верхнего порога.

Попробуйте проверить данное утверждение, поставив сначала порог 100,200, а затем 10,20. Вы увидите, насколько больше границ на экране.



Также заметьте, насколько меньше стало FPS, камера работает приблизительно в 2-3 раза медленнее.

Занятие 5. Распознавание сложных объектов.

Преимущество OpenMV в том, что большая часть сложностей Python и основной библиотеки машинного зрения (OpenCV) скрыты для пользователя, оставляя лишь понятные четкие функции. Однако нам необходимо понимать, как именно они работают.

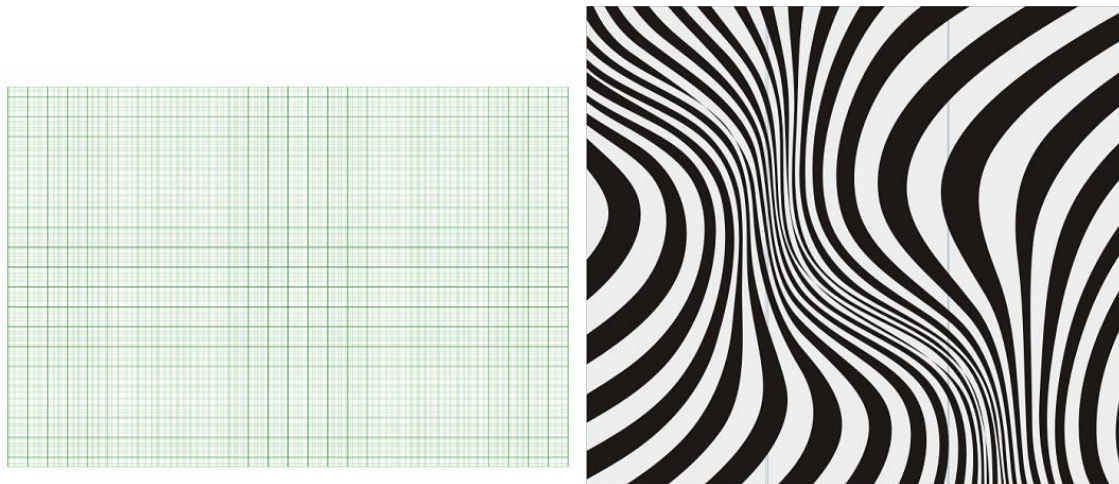
Для распознавания простых геометрических объектов (линии, круги, эллипсы) используется обобщенное преобразование Хафа (Hough transform).

Этот алгоритм запатентован Полом Хафом в 1962, и доработан группой ученых позднее, в 1981 году.

Здесь и далее, вы должны понимать, что те функции, которые мы пишем одним словом, имеют под собой многолетний академический труд, и непременно возникнут сложности с точным пониманием происходящего.

Если не залазить в фазовые пространства и всю сложную алгебру, то давайте возьмем за истину тот факт, что любой простой объект можно представить в виде какой-либо формулы (в разных системах отсчета, конечно). Как пример, любая прямая имеет формулу $y = kx + b$, так что, чтобы задать одну и ту же прямую в декартовых координатах, нам нужно знать только два параметра: k и b .

Дальше, учтем тот факт, что каждый пиксель на изображении имеет только два состояния: он входит в прямую (или квадрат, круг и тд) или не входит в нее. Для компьютера это самый удобный вариант, 1 или 0. Но, шанс того, что пиксель входит в прямую или не входит, абсолютно разный, сравните два фото ниже.



На фото с миллиметровой бумагой прямых будет явно больше, поэтому шанс того, что любой пиксель изображения будет входить в состав прямой, явно выше. Этот термин в математике называют «вес».

Так вот, каждый пиксель является часть какого-либо набора прямых. И алгоритм Хафа прогоняет каждый пиксель, сравнивает их значения параметров, и если у соседних пикселей значения достаточно близки (а это проверяется с помощью некоей весовой функции), то алгоритм возвращает функцию данной прямой.

Далее эти прямые необходимо отрисовать для наглядности, с помощью уже известных нам функций.

Откроем пример «find_circles.py» из папки «Feature Detection». Первые 18 строк уже известны из предыдущих глав.

Однако теперь мы получаем изображение не через *sensor.snapshot()*, но добавляем *lens_corr(1.8)*. Данный параметр необходим, чтобы убрать возможные эффекты «рыбьего глаза», так как камера OpenMV обладает повышенной дисторсией для увеличения угла обзора (дисторсия – вид преломления на оптических системах, при котором изображение изменяется линейно по мере удаления от оптической оси). Для того, чтобы лучше понять возможности компьютерной обработки изображения, удалите на время все после 19 строки, и попробуйте увеличить параметры на 0.5, а затем 4.



Значение коррекции – 3.

Для камеры OpenMV оптимальное значение коррекции – 1.8, так что поставим его обратно.

Далее прописан счетный цикл, перебирающий все данные, которые вернет *img.find_circles*. У данной функции большое количество параметров, что позволяет точно настроить алгоритм по необходимым критериям. Прописаны несколько из них – *x,y,r_margin* – максимальное расстояние между схожими точками, чтобы просуммировать их в единый круг. *R_min/max* – минимальный и максимальный радиусы кругов, соответственно.

Find_circles возвращает массив данных, который необходимо отрисовать на данной изображении, для наглядности выполнения алгоритма. Для этого используется функция *img.draw_circle*, в атрибутах которой прописаны координаты круга, его радиус, цвет и толщина.

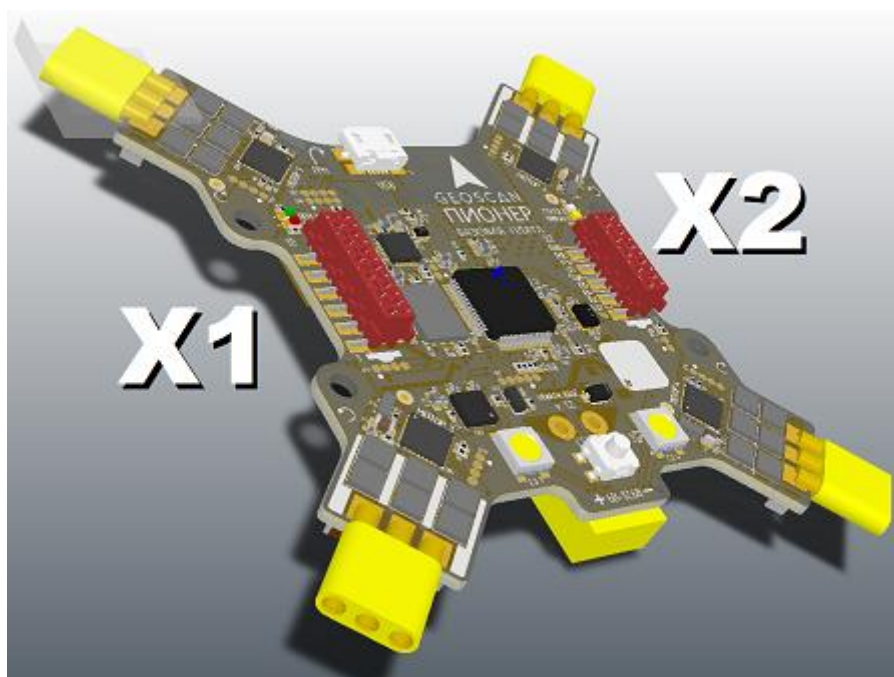


Как мы видим, алгоритм не идеально точен, так как использовались стандартные параметры внутри `find_circles`.

Занятие 6. Подключение OpenMV к полетной плате Пионер, протоколы связи. Программирование камеры для полета по ArUco меткам.

Перед подключением камеры OpenMV необходимо разобраться, что именно мы можем подключать к коптеру, и каким образом оно подключается.

На полетной плате находятся два основных шлейфа для подключения периферии – X1 и X2.



Распиновка шлейфов находится в документации.

Периферия может подключаться любым доступным способом – SPI, USART, ADC, PWM. Для общего понимания необходимо поговорить о каждом из них.

SPI (Serial peripheral interface) – последовательный периферийный интерфейс, самый популярный и оптимальный вариант подключения периферии. Характеризуется полной дуплексной передачей данных (двунаправленная передача, как говорить по телефону), высокой пропускной способностью, произвольной длиной пакета, а также низким энергопотреблением и малым количеством выводов.

Вместе с этим SPI имеет некоторые минусы, основной – отсутствие подтверждения приема данных, невозможность ведомого устройства управлять потоком данных.

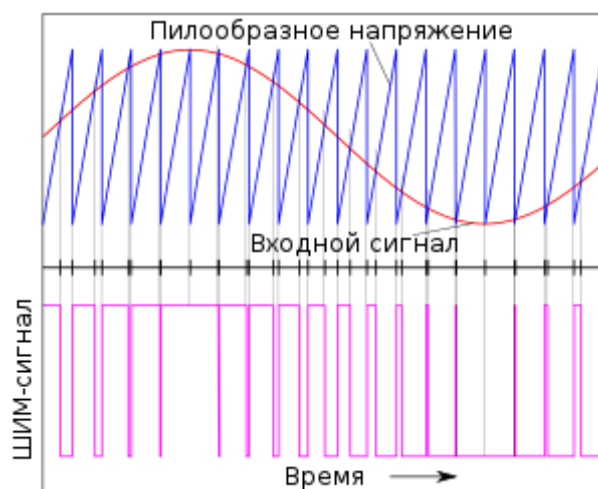
В SPI используются 4 цифровых вывода:

- SCLK – последовательный тактовый сигнал, необходим для передачи ведомому устройству тактового сигнала
- MOSI | MISO (Master in/out Slave out/in) – передача данных в обе стороны
- CS – выбор ведущего и ведомого (Master / Slave)

ADC (Analog-to-digital converter) – аналогово-цифровой преобразователь.

Один из самых старых методов, так как исторически большая часть оборудования была аналоговая. Для пояснения, аналоговый сигнал – непрерывен по времени, цифровой – дискретный, т.е. состоит из множества отдельных значений. АЦП работает с помощью квантования сигнала (т.е. разбиение сигнала на некоторые величины с определенной частотой).

PWM (Pulse-width modulation) – широтно-импульсная модуляция. Достаточно популярный метод, переводящий сигнал в ШИМ.



Про принцип работы USART рассказывалось в предыдущем методическом пособии, так что не буду заострять на этом внимание.

В прошлом пособии мы рассматривали кейс с изменением яркости свечения светодиодов в зависимости от освещенности, которую видит камера. Однако этот кейс не сильно применим ко всему, что может быть связано с коптерами, так что не будем его повторять.

Единственный на данный момент кейс, который задействует камеру OpenMV и Пионер базовый – распознавание ArUco меток, и дальнейшее действие в зависимости от метки. Это объемный и сложный кейс, поэтому он будет разбит на два занятия.

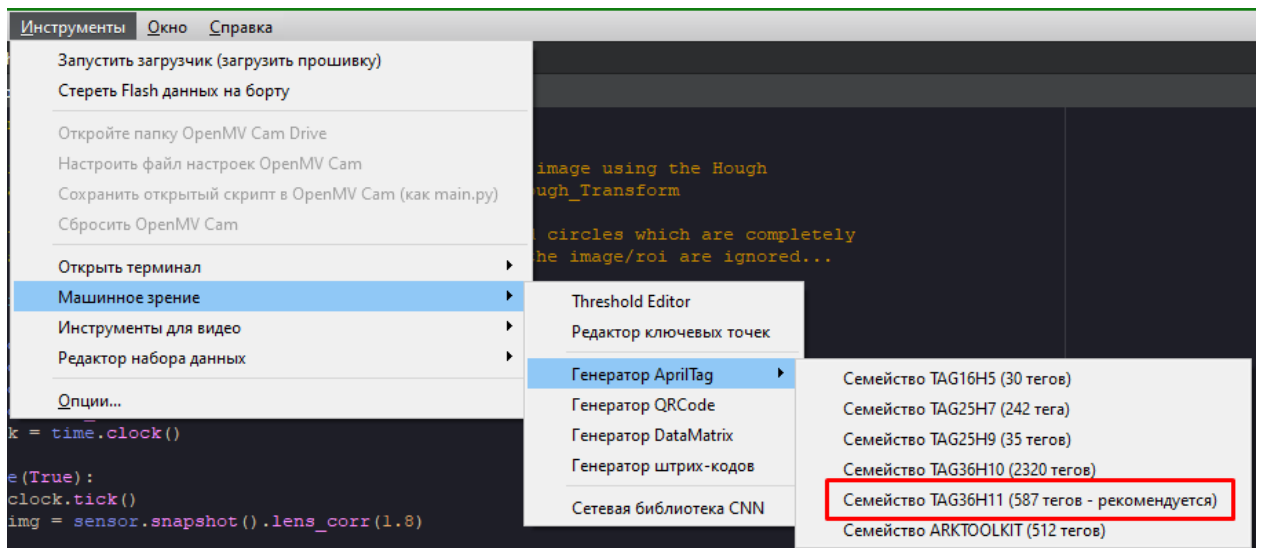
Наверняка все из вас знают, что такое QR-код. Это некая последовательность данных, закодированная в виде (чаще всего) черно-белого квадратичного изображения. Изначально QR-коды создавались именно в автомобильной промышленности для более быстрого распознавания детали.

В нынешнее время QR-коды разных видов очень удобны для автономной техники, так как они однозначно и быстро считываются. Для камеры OpenMV предпочтительны ArUco маркеры – некая разновидность QR кодов, которые содержат в себе только один байт данных – число.

Соответственно, нашей задачей будет получить ArUco маркеры, запрограммировать камеру OpenMV для распознавания этих маркеров, а также запрограммировать Пионер базовый для того, чтобы коптер делал различные части программы в зависимости от того, какую метку он увидел.

Итак, для начала необходимо получить метки, которые будем распознавать.

Заходим во вкладку «Инструменты», далее «машинное зрение» и выбираем нужный класс AprilTag.



Около каждого семейства расписано число различных тегов, которые будут в метке.

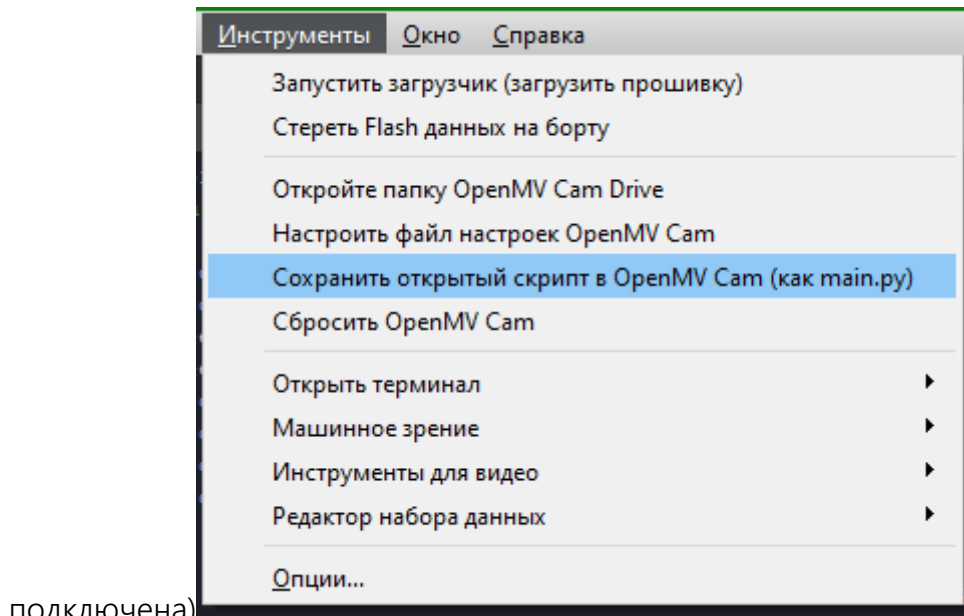
После этого выбираем число тегов, и папку, куда они сохранятся (в виде изображений).

До этого мы лишь запускали скрипт посредством OpenMV IDE, не загружая код на камеру, однако при подключении камеры к Пионеру, скрипт запускается автоматически после подачи питания.

Варианта загрузки скрипта на камеру два:

1. На самой камере уже существует скрипт `main.py`, который можно открыть из любого текстового редактора или IDE (собственно, с помощью OpenMV IDE, ну либо блокнотом) и менять напрямую.

2. Либо, после создания скрипта, необходимо зайти во вкладку «Инструменты», «Сохранить скрипт ...» (естественно, для этого камера должна быть



подключена)

`../main.py`

Программа начинается с всех импортов, это различные используемые библиотеки и отдельно библиотеки работы с UART/LED.

```
1 import sensor, image, time, math, pyb
2 from pyb import UART, LED
3
4 sensor.reset()
5 sensor.set_pixformat(sensor.RGB565)
6 screen_w = 160
7 screen_h = 120
8 sensor.set_framesize(sensor.QQVGA)
9 sensor.skip_frames(30)
10 sensor.set_auto_whitebal(False)
```

Далее выполняется настройка камеры, выставляем цветовое пространство RGB565, разрешение камеры выставляем 160x120 (из практики, это оптимальное разрешение для быстрого действия камеры).

Далее выполняем пропуск 30 первых кадров камеры для выполнения настройки, а также отключаем автоматический баланс белого.

Далее необходимо прописать UART. Его будем прописывать и со стороны копитера, и со стороны камеры. Не удивляйтесь разным номерам UART, это обычная

прихоть производителя (у Пионера UART имеет номера 1 и 4, у нашей версии OpenMV только один UART – 3).

```
12  uart = UART(3)
13  uart.init(9600, bits=8, parity=None, stop=1, timeout_char=1000)
14
```

Первый параметр – baudrate. Бод – единица измерения символьной скорости (что не совсем то же самое, что и скорость передачи данных), названная в честь французского инженера Эмиля Бодо. Бодрейт равен скорости передачи данных только в случае двоичной кодировки, что в современной технике практически не используется. За каждый бод передается от 4 до 16 бит информации. Следовательно, при бодрейте 9600, камера передает в среднем $9600 \times 4 = 38400$ бит/с или ~4,6 кБ/с. Это звучит крошечными числами в мире, где используются гигабитные каналы связи, однако следует помнить, что и камера, и полетная плата – достаточно маломощная техника.

Остальные параметры – число бит в каждом пакете, есть ли бит четности, сколько бит используются в качестве стоп-бита, и первоначальный таймаут передачи данных.

После инициализируем все цвета светодиода отдельными переменными, для удобства.

```
15  red_led = pyb.LED(1)
16  green_led = pyb.LED(2)
17  blue_led = pyb.LED(3)
18
```

Далее необходима функция, которая будет передавать необходимые данные по UART.

В параметрах этой функции будут несколько переменных – id метки, и координаты от центра камеры до центра метки, т.е. 3 параметра.

```

22 def sendPacket(ledState, dx, dy):
23     uart.writechar(0xBB) # packet begin byte
24     uart.writechar(ledState)
25     uart.writechar(dx.to_bytes(1, 'big')[0])
26     uart.writechar(dy.to_bytes(1, 'big')[0])
27     uart.writechar(0xFF) # packet end byte
28

```

В теле функции 5 команд, все они передают что-либо по UART. Первая и последняя – начало и конец пакета данных, соответственно. Они не меняются (0xBB, 0xFF). Между ними идет запись данных. Примечательно, что dx и dy будут float тип данных (дробные числа), соответственно их надо особым образом перевести в побитовые значения с помощью команды to_bytes. Первый параметр этой функции – число байтов, а второй – порядок записи (если big, то самый старший байт будет записываться слева, если little, то справа).

Далее прописываем весь основной блок кода.

```

29 while True:
30     clock.tick()
31     img = sensor.snapshot()
32     #img.rotation_corr(z_rotation = 90)
33     apriltag_array = img.find_apriltags().sort(key=lambda x: x.id())
34
35     if len(apriltag_array) == 0:
36         red_led.on()
37         green_led.off()
38         blue_led.off()
39         sendPacket(0, 0, 0) # send info, no qr code found
40     else:
41         for tag in apriltag_array:
42             img.draw_rectangle(tag.rect(), color = (255, 0, 0))
43             img.draw_cross(tag.cx(), tag.cy(), color = (0, 255, 0))
44             red_led.off()
45             green_led.off()
46             blue_led.on()
47             print_args = (tag.id(), (180 * tag.rotation())/math.pi)
48             print("Tag Number", str(tag.id()))
49             dx = int(tag.cx() - screen_w/2)
50             dy = int(tag.cy() - screen_h/2)
51             print("dx=" + str(dx) + ", dy=" + str(dy))
52             sendPacket(tag.id(), dx, -dy)
53             break
54     print(clock.fps())
55

```

Создаем бесконечный цикл, который будет искать AprilTags (строка 33). По умолчанию данная функция распознает семейство TAG36H11, если вы используете другое, это необходимо прописать (family="<название семейства>").

Далее необходима индикация, распознала ли камера метку. Для этого используется конструкция `if/else`, где мы включаем красный светодиод и отправляем пустой пакет по UART.

Если же метка найдена, код идет в блок `else`.

Так как `apriltag_array` – массив, нам необходимо раскрыть его с помощью счетного цикла `for`.

Первые две команды этого цикла отрисовывают на экране красный квадрат вокруг метки и зеленый крест на ее центре.

Далее сигнализируем светодиодом то, что метка найдена (в моем случае это свечение синим цветом) и выводим в консоль номер найденного тега.

После этого необходимо высчитать расстояние от центра метки до центра экрана, и также вывести это в консоль.

Последняя команда отправляет пакет с данными по UART, однако заметьте, что координата по `dy` отправляется с минусом, так как направление движения коптера инвертировано с направлением камеры.

Занятие 7. Программирование квадрокоптера для полета по ArUco меткам.

После завершения кода для камеры OpenMV, необходимо подготовить скрипт для самого квадрокоптера.

Основной посыл полета квадрокоптера в том, что после взлета мы ищем метку, а далее взаимодействуем с ней, в данном уроке мы будем менять направление полета.

Ограничение заключается в том, что мы не можем прервать полет к точке по ходу движения, из-за этого необходимо передвижение маленькими шагами до момента нахождения точки.

В данном занятии мы воспользуемся более простым вариантом, задавая направление полета с помощью таймера и пары флагов.

Начнется наш код с инициализации светодиодов, а также функции, переключающей их цвет.

```

1 local ledNumber = 4
2
3 local leds = Ledbar.new(ledNumber)
4
5 local colors = {
6     {0.1, 0, 0}, -- красный
7     {0.1, 0.1, 0.1}, -- белый
8     {0, 0.1, 0}, -- зеленый
9     {0.1, 0.1, 0}, -- желтый
10    {0.1, 0, 0.1}, -- фиолетовый
11    {0, 0, 0.1}, -- синий
12    {0, 0, 0} -- черный/отключение светодиодов
13 }
14
15 local unpack = table.unpack
16
17 function changeColor(color)
18     -- проходим в цикле по всем светодиодам с 0 по 3
19     for i=0, ledNumber - 1 do
20         leds:set(i, unpack(color))
21     end
22 end

```

После этого необходимо расписать подключение по UART, чтобы получать данные с камеры.

Если посмотреть распиновку платы, у нее есть UART №1 и №4, использовать мы будем 4. Бодрейт 9600, 8 бит данных, 1 стоп-бит и нет бита четности.

```

25
26 local uartNum = 4 -- номер UART (на коптере их два: 1 и 4)
27 local baudRate = 9600 -- скорость передачи данных
28 local dataBits = 8 -- количество принимаемых бит
29 local stopBits = 1
30 local parity = Uart.PARITY_NONE -- бит чётности
31 local uart = Uart.new(uartNum, baudRate, parity, stopBits) -- создание порта управления UART-ом

```

Далее пропишем несколько функций для получения данных по инициализированному UART.

Первые две будут отвечать за чтение одного пакета по UART и распознавание каждого бита данных.

```

33 function getc() -- функция чтения пакета
34 while uart:bytesToRead() == 0 do
35 end
36 return uart:read(1) -- чтение одного бита
37 end
38
39
40 function ord(chr, signed) -- функция распаковки и опознавания битов (знаковые или беззнаковые)
41 local specifier = "B"
42 if signed then
43 specifier = "b"
44 end
45 return string.unpack(specifier, chr)
46 end

```

Функция `getc` по очереди читает каждый бит данных, пока эти данные есть. Функция `ord` необходима для распознавания знаковых и беззнаковых битов.

Далее прописываем функцию распаковки пакета данных и обработки данных. Как мы помним из предыдущего урока, камера отправляет 3 переменные – номер тэга и его смещения относительно центра камеры.

```

48 function getData() -- функция побитовой распаковки
49 while true do
50 if (ord(getc()) == 0xBB) then
51 break
52 end
53 end
54 local ledstate = ord(getc()) -- считывание id метки
55 local dx = ord(getc(), true) -- считывание смещения метки по dx относительно центра камеры
56 local dy = ord(getc(), true) -- считывание смещения метки по dy относительно центра камеры
57 ord(getc()) -- считывание конца пакета
58 return ledstate, dx, dy
59 end

```

Соответственно, именно эти данные в том же порядке и считываем. `0xBB` это конец пакета данных, `ledstate`, `dx`, `dy` – используемые далее переменные.

Далее необходимо создать скрипт полета в точку. Так как летаем в ОРТ, точка 0,0 – точка взлета коптера. И после каждого перемещения необходимо записывать новые координаты, чтобы `goToLocalPoint` работал корректно.

```

62 height = 0.7 -- высота полета
63
64
65 xCord = 0
66 yCord = 0
67
68 function new_point(x,y) -- полет в новую точку
69     xCord = xCord + x
70     yCord = yCord + y
71     ap.goToLocalPoint(xCord,yCord,height)
72 end
73

```

Теперь осталось только создать функцию поиска метки, создать таймер и запустить его.

Летаем мы в 4 направлениях, соответственно нужны 4 флага движения (назову их по направлениям – forw,back,left,right).

Эти флаги имеют значения true или false, т.е. если стоит флаг forw = true, то коптер должен лететь вперед, если right = true, то коптер должен лететь вправо. Полет по диагонали не прописывался, уверен, что вы допишете его по аналогии в случае необходимости.


```

77 forw = false -- флаги движения
78 right = false
79 left = false
80 back = false
81
82 function finding() -- функция поиска метки
83     if (mark == 1) then
84         changeColor(colors[1])
85         forw = true
86         right = false
87         left = false
88         back = false
89     end
90     if (mark == 2) then
91         changeColor(colors[2])
92         forw = false
93         right = true
94         left = false
95         back = false
96     end
97     if (mark == 3) then
98         changeColor(colors[3])
99         forw = false
100        right = false
101        left = true
102        back = false
103    end
104    if (mark == 4) then
105        changeColor(colors[4])
106        forw = false
107        right = false
108        left = false
109        back = true
110    end
111 end

```

В данном примере я использую тэги от 1 до 4, где 1 – лететь вперед, 2 – лететь вправо, 3 – лететь влево и 4 – лететь назад.

Номера тэгов прописываются в конструкции if/else, mark == N. Также, каждое направление полета будет использовать собственную светодиодную индикацию для понимания текущего состояния полета.

Далее остается создать таймер, который будет получать данные и задавать направление полета.

В нем мы прописываем получение данных из функции с UART. Также пропишем relative x/y, для будущего самостоятельного выполнения.

Внутри таймера используем наши флаги, которые будут задавать направление полета. Шаг полета зададим минимальный, чтобы коптер каждые несколько сантиметров проверял, не нашел ли он новую метку.

```
116 ▾ goT = Timer.new(0.1, function() -- основной таймер
117     mark, dx, dy = getData() -- получение данных с камеры каждые 0.1 с
118     relative_x = dx/600 --примерный перевод пикселей камеры в метры для полёта
119     relative_y = dy/600
120     finding() -- поиск метки
121 ▾     if forw then
122         new_point(0, 0.04)
123     end
124 ▾     if right then
125         new_point(0.04, 0)
126     end
127 ▾     if left then
128         new_point(-0.04, 0)
129     end
130 ▾     if back then
131         new_point(0, -0.04)
132     end
133 end)
134
```

Осталось не забыть функцию callback(event), без которой коптер не взлетит, а также прописать стандартные команды для прогрева двигателей и взлета (и запуск таймера, конечно).

```
136 ▾ function callback(event)
137     end
138
139     -----
140
141     changeColor(colors[6])
142     ap.push(Ev.MCE_PREFLIGHT)
143     sleep(3)
144     ap.push(Ev.MCE_TAKEOFF)
145     ap.goToLocalPoint(0,0,height)
146     goT:start()
147
```

Теперь наш коптер умеет распознавать несколько меток и летать в разных направлениях. Для корректной работы кода коптер должен взлетать с какой-либо метки, чтобы сменился флаг направления полета.

Занятие 8. Контрольный рубеж.

В предыдущем занятии мы дописали код, использующий камеру OpenMV для полета по тэгам.

Если вы запускали его, то наверняка заметили, что коптер меняет направление сразу же, как видит метку, что иногда приводит к погрешностям полета, и иногда он может пролететь следующую метку.

Задание этого урока состоит в том, чтобы прописать любой механизм центровки, используя переменные `relative x/y`, чтобы минимизировать данную погрешность.

И задание это, конечно же, самостоятельное.



Методическое пособие является интеллектуальной
собственностью ООО «Геоскан».
2021