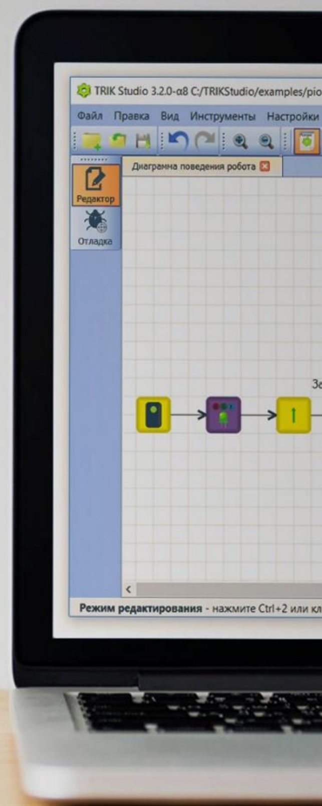
ФОНД СОДЕЙСТВИЯ РАЗВИТИЮ
ВОЕННОГО ОБРАЗОВАНИЯ



GEOSCAN

«Робототехника и управление беспилотными авиационными системами»

Авторы-составители — Образовательная команда Geoscan:
Азибаев Р.С., Грибова Л. А.

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

Учебный план 4

Введение 5

Занятие 1. Знакомство со средой программирования TRIK Studio 6

Занятие 2. Знакомство со средой Pioneer Station 10

Занятие 3. Программирование в TRIK Studio – управление светодиодами 12

Занятие 4. Программирование на языке Lua – управление светодиодами. 14

Занятие 5. Автономная навигация БАС – Оптическое позиционирование 18

Занятие 6. Автономная навигация БАС – Маяки. 21

Занятие 7. Автономная навигация БАС – Точность 24

Занятие 8. Объединение пилотируемого полета и автономной программы 25

Занятие 9. Работа с автопилотом 28

Занятие 10. Соревнование «Скорая помощь» 33

Занятие 11. Программирование на языке Lua – Таблицы 35

Занятие 12. Программирование на языке Lua – Циклы. 42

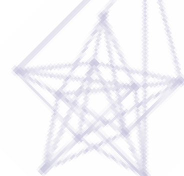
Занятие 13. Программирование на языке Lua – Управление светодиодной панелью 47

Занятие 14. Программирование на языке Lua – Таймеры и асинхронность 52

Занятие 15. Класс Sensors. Получение данных от автопилота. 58

Занятие 16. Программирование на Lua – «Формула 1» 60

Занятие 17. Программирование на языках Lua и Python – Программируемая камера OpenMV. 63



ФОНД СОДЕЙСТВИЯ РАЗВИТИЮ ВОЕННОГО ОБРАЗОВАНИЯ

Учебный план

№	Тема	Всего часов
1	Вводное занятие. Повторение пройденного материала предыдущего модуля.	2
2	Модуль 1. Теоретические основы программирования, знакомство со средами разработки, введение в блочное программирование. Эксплуатация БАС, основы работы прошивок, обучение работе с микроконтроллерами	10
3	Модуль 2. Начало работы с модулями квадрокоптера «Пионер», объяснение работы адресных светодиодов, работа с автономными программами в рамках ручного пилотирования	13
4	Модуль 3. Описание систем позиционирования БАС на примере квадрокоптера «Пионер». Программирование автономного полета в различных системах позиционирования. Работа с автопилотом, отладка кода	13
5	Промежуточный контроль знаний по модулям 1-3.	2
6	Модуль 4. Описание высокоуровневых языков программирования Lua и Python. Основные принципы работы кода, ООП, управление различными модулями с учетом ООП.	14
7	Модуль 5. Работа с автопилотом, получение данных с пульта ДУ, ветвление кода в зависимости от пульта ДУ	8
8	Промежуточный контроль знаний по модулям 4-5	2
9	Модуль 6. Введение в машинное зрение, основные библиотеки для работы с камерами машинного зрения. Программирование камеры OpenMV на языке Python	8
	Всего	72

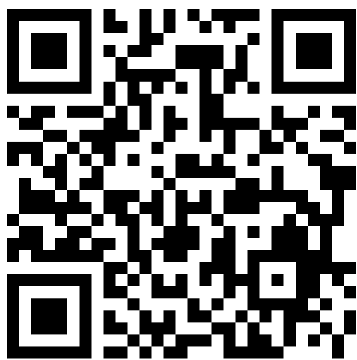
ФОНД СОДЕЙСТВИЯ РАЗВИТИЮ ВОЕННОГО ОБРАЗОВАНИЯ

Введение

Данное методическое пособие предназначено для образовательных учреждений, авиамодельных секций, кружков робототехники, или самостоятельного изучения.

Целью пособия является обучение основам программирования и автономному полету БПЛА.

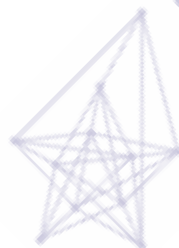
Большая часть представленных примеров выложена в электронном виде:



Код, используемый в пособии выделен курсивом:

local led_number = 4

Перед началом работы необходимо скачать IDE: TRIK Studio, Pioneer Station, Geoscan LPS и OpenMV IDE. Все ссылки есть в документации Пионера и на GitHub.



ФОНД СОДЕЙСТВИЯ РАЗВИТИЮ ВОЕННОГО ОБРАЗОВАНИЯ

Занятие 1. Знакомство со средой программирования TRIK Studio

Теория:

Визуальное программирование – технология программирования, которая предусматривает создание программ с помощью визуальных блоков кода.

Одним из вариантов программирования Пионера является программа TRIK Studio (рис.1).

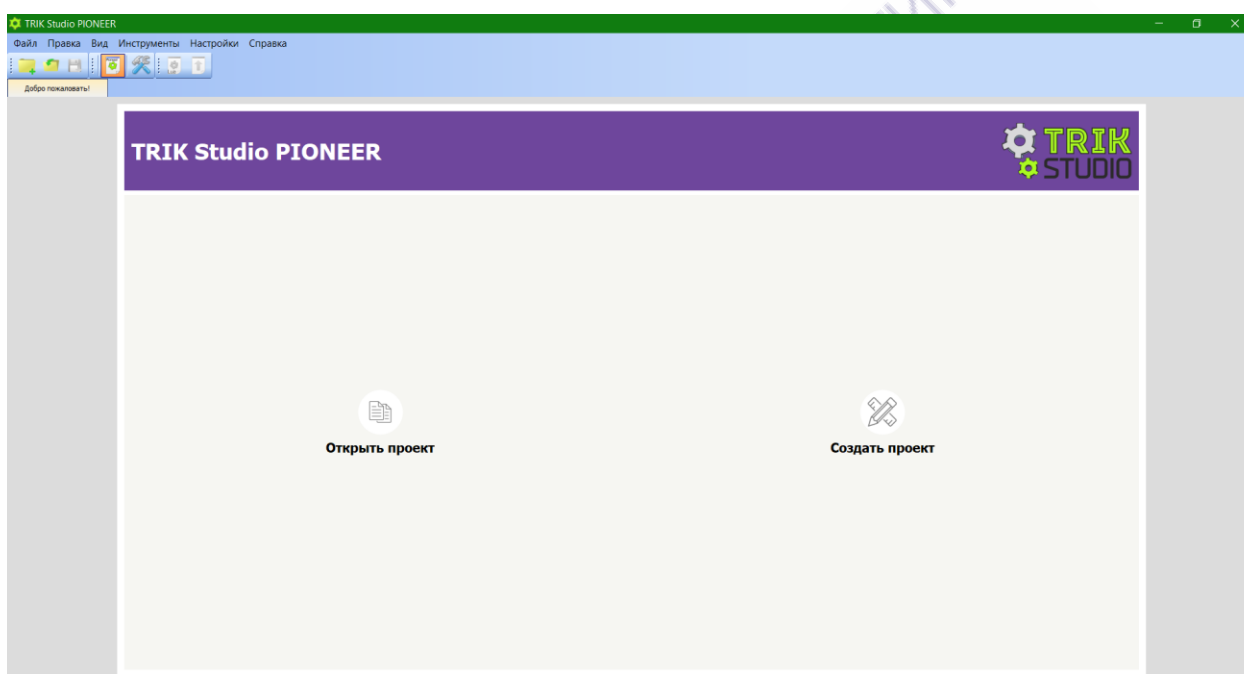


Рис.1 Начальное окно программы

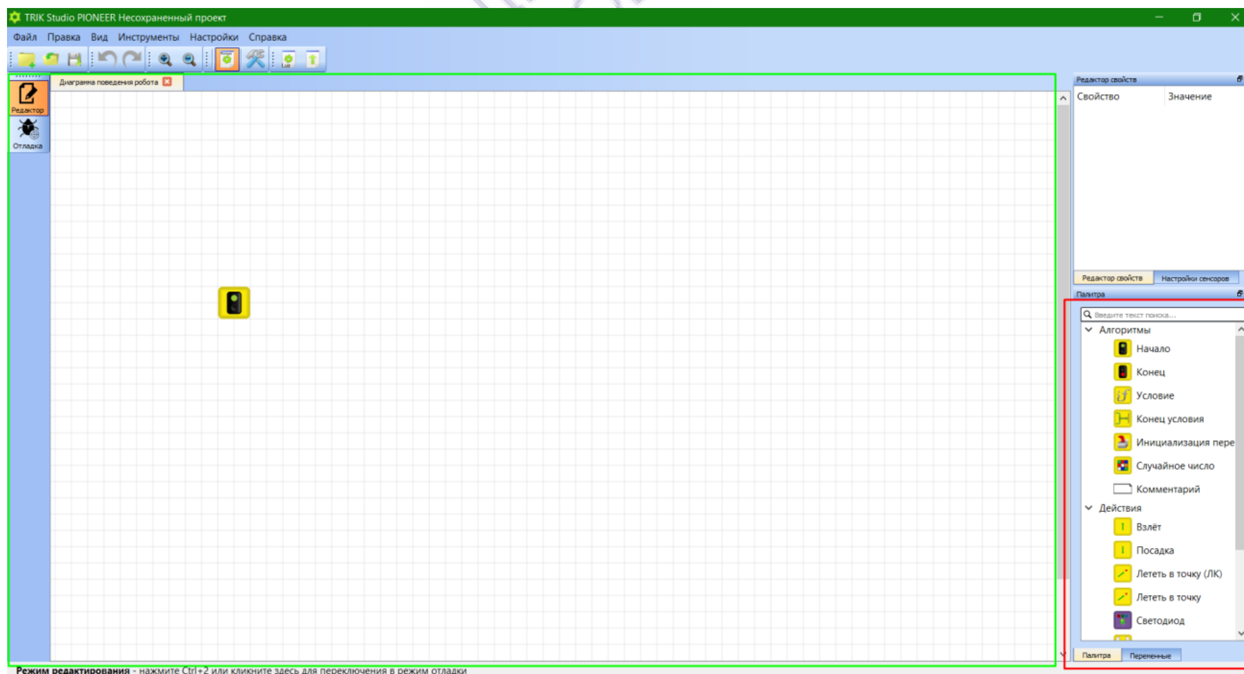


Рис.2 Проект в TRIK Studio

После нажатия «Создать проект» видим два основных блока (рис.2). Красным справа указан список блоков, которые можем использовать при работе. Зеленым, соответственно, поле для размещения этих блоков.

Все связи между блоками создаются ПКМ, каждый блок редактируется нажатием на него.

Обязательными блоками являются «Начало» и «Конец».

Практика:

1. Первая программа в TRIK Studio, работа со светодиодом.

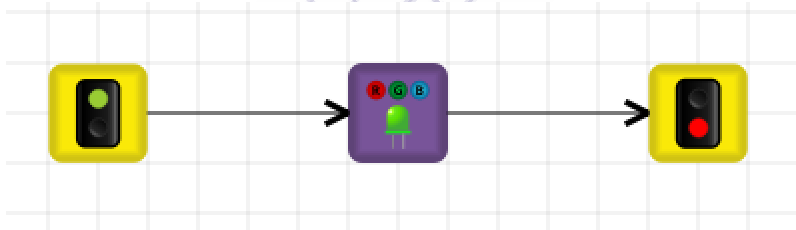
На главной плате Пионера находятся 4 светодиода, пронумерованные от 0 до 3.

Каждый светодиод может светить в любом цвете (о том, как настроить цвета и яркость светодиодов мы поговорим на следующих занятиях).

Возьмём с панели справа блок «Светодиод» и поставим его между блоками «Начало» и «Конец».

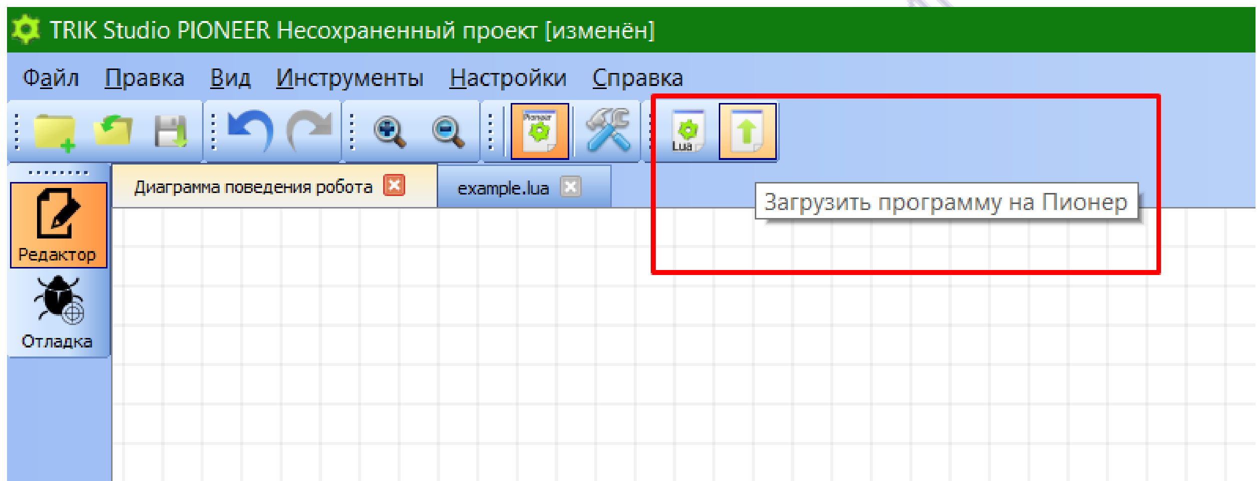
ФОНД СОДЕЙСТВИЯ РАЗВИТИЮ ВОЕННОГО ОБРАЗОВАНИЯ

Правой кнопкой мыши соединим блоки.



При нажатии на блок светодиода выставим «красный» равным 1, остальные значения оставим 0. Таким образом, после загрузки программы светодиод под номером 0 будет гореть красным цветом.

После этого нажимаем на «загрузить программу на Пионер»



После нажатия автоматически откроется Pioneer Station.

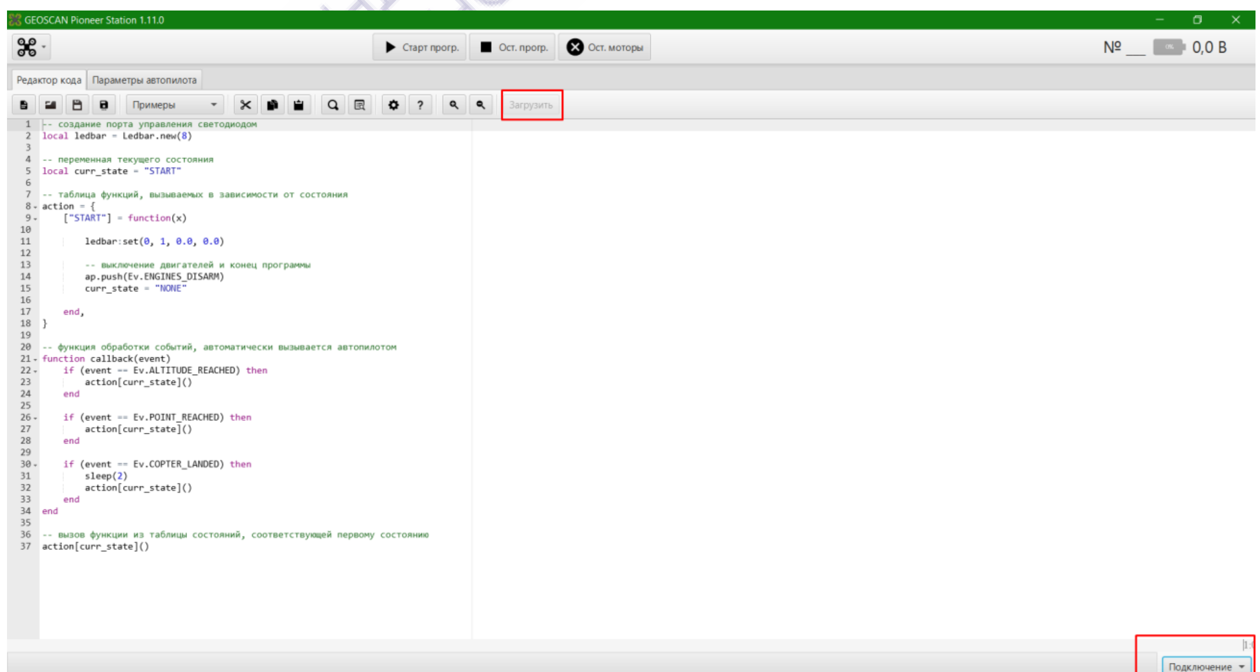


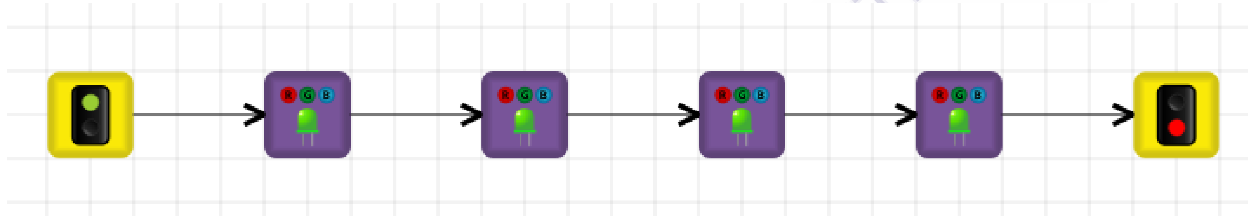
Рис. 3 Рабочее окно Pioneer Station

Этот код мы пока что не трогаем, подключаем Пионер по USB, снизу справа нажимаем «Подключение по кабелю USB», и загрузить (выделено красным).

После этого, при нажатии на кнопку «Старт» светодиод под номером 0 загорится красным.

2. Самостоятельная работа

Попробуйте самостоятельно узнать нумерацию светодиодов, подставляя в код разные номера светодиодов и цвета (любые значения от 0 до 1).



3. Работа с блоком «Таймер».

Найдем на боковой панели блок «Таймер» и поставим их на рабочее поле, как на рисунке 4.

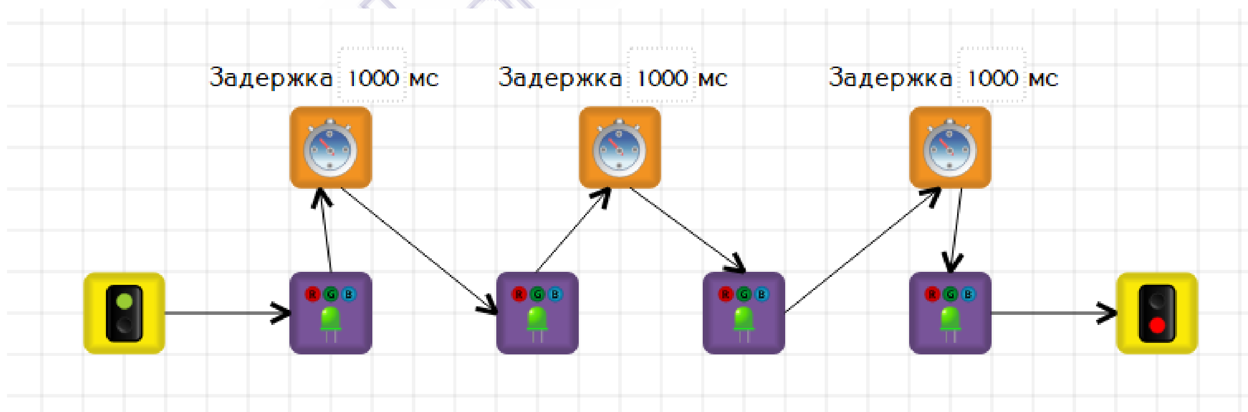


Рис. 4 Схема блоков «Таймер»

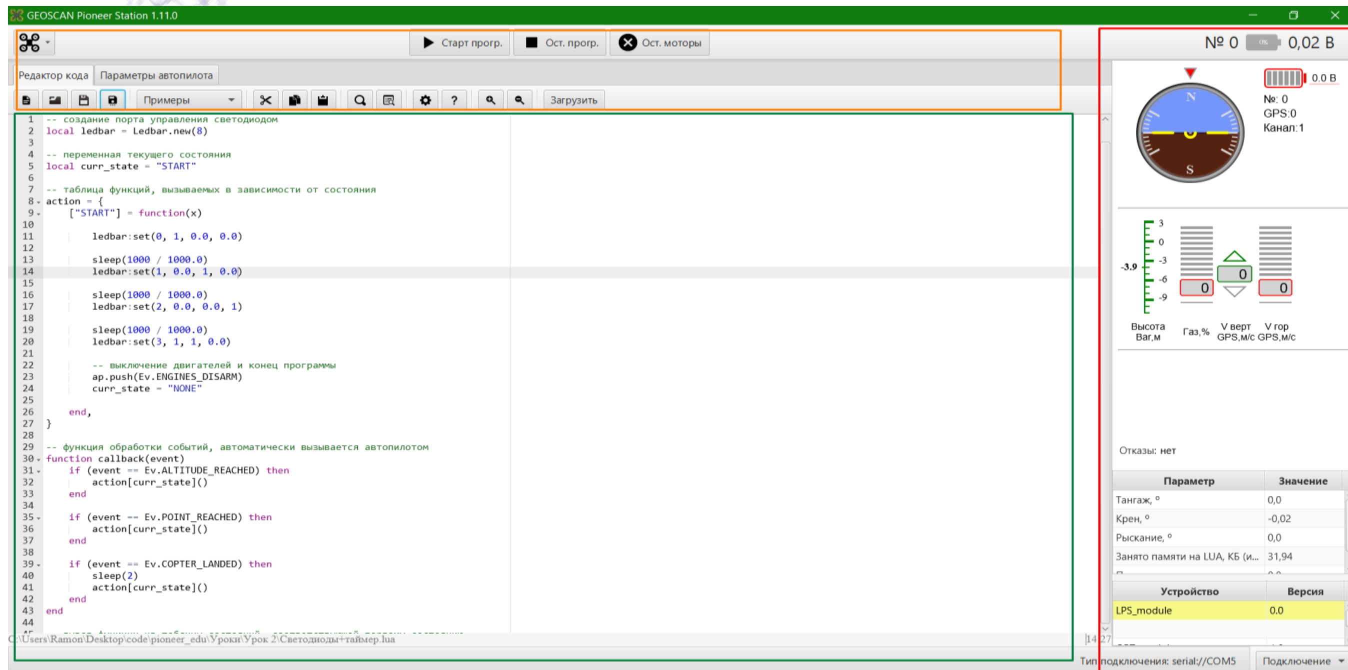
Данный блок позволяет сделать паузу в выполнении программы. Величину паузы указываем самостоятельно.

4. Самостоятельная работа

Занятие 2. Знакомство со средой Pioneer Station

Теория:

Блочное программирование просто для понимания, однако TRIK Studio не позволяет использовать функционал дрона полностью. Основной средой для программирования Пионера является Pioneer Station.



Красный блок справа появляется при подключении Пионера, в нем можно увидеть информацию с датчиков Пионера.

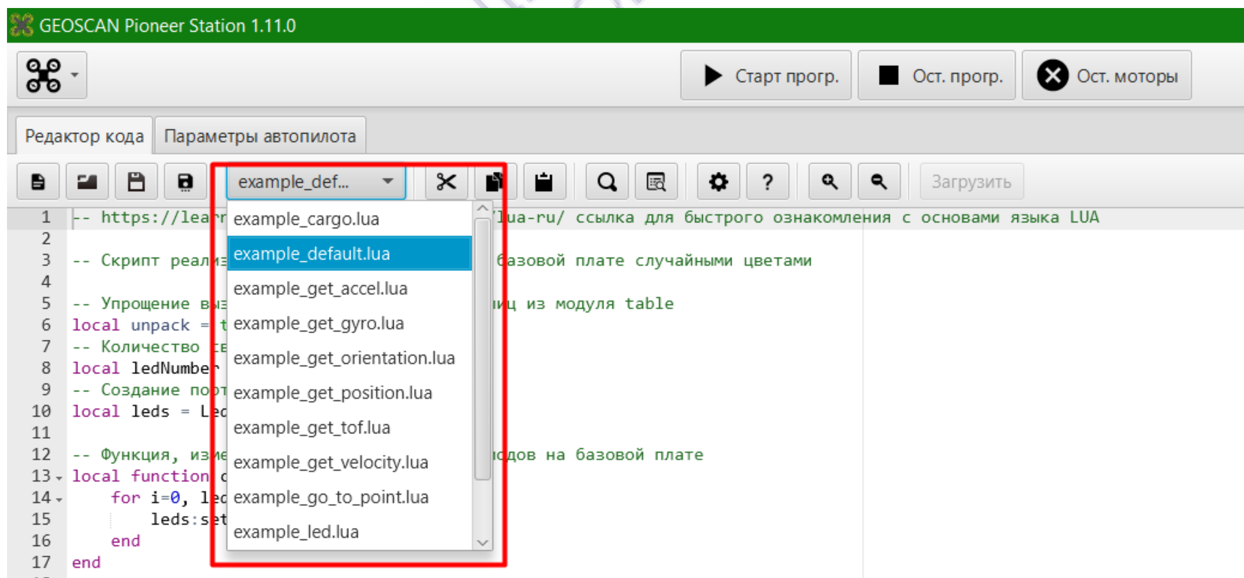
Зеленой рамкой выделено рабочее пространство студии.

Пионер написан на языке Lua, краткую информацию о нем вы можете почитать по ссылке из GitHub.

Практика:

1. Примеры программ.

В Пионере прописаны несколько примеров. Запустим некоторые из них. Найти их можете на скриншоте.



Каждый пример хорошо документирован, попробуйте запустить некоторые из них, допустим, *example_default.lua*, который выполняет мигание светодиодов на плате.

Вот список всех примеров:

1. *Example_cargo* – включение/выключение магнита каждую секунду
2. *Example_default* – мигание светодиодами на плате
3. *Example_get_*** - получение различной информации о полете и вывод ее на светодиоды
4. *Example_go_to_point* – автономное перемещение дрона по заданным точкам
5. *Example_led* – вывод чисел от 0 до 10 разными цветами.
6. *Example_path* – усложненный вариант автономного полета
7. *Learnlua* – файл, показывающий все возможности языка Lua

Для выполнения некоторых примеров необходимы дополнительные модули, такие как модуль подъема груза или led панель

Большую часть примеров мы напишем самостоятельно на будущих уроках, последовательно улучшая навыки программирования Пионера.

Занятие 3. Программирование в TRIK Studio – управление светодиодами

Теория:

Азбука Морзе – способ кодирования алфавита, цифр и знаков препинания с помощью двух знаков: точки и тире.

На рисунках показаны варианты английской и русской азбук Морзе.

С изобретением электрического телеграфа в 1774 году, требовалось придумать некий код, позволяющий только по длительности электрического сигнала шифровать и передавать различные символы. На протяжении 70 лет этот код модернизировался, пока в 1838 году Морзе не предложил свой вариант.

Справедливости ради, данный вариант также дорабатывался на протяжении десяти лет. Наибольший вклад внес Фридрих Герке, и этим вариантом мы пользуемся до сих пор.

Вслед за английской морзянкой стали развиваться азбуки Морзе на других языках, большинство из них были придуманы в течение пары лет после английской. Наибольший вклад в развитие телеграфа в России сделал Борис Якоби, русский физик-изобретатель.

ФОНД СОДЕЙСТВИЯ РАЗВИТИЮ ВОЕННОГО ОБРАЗОВАНИЯ

Практика:

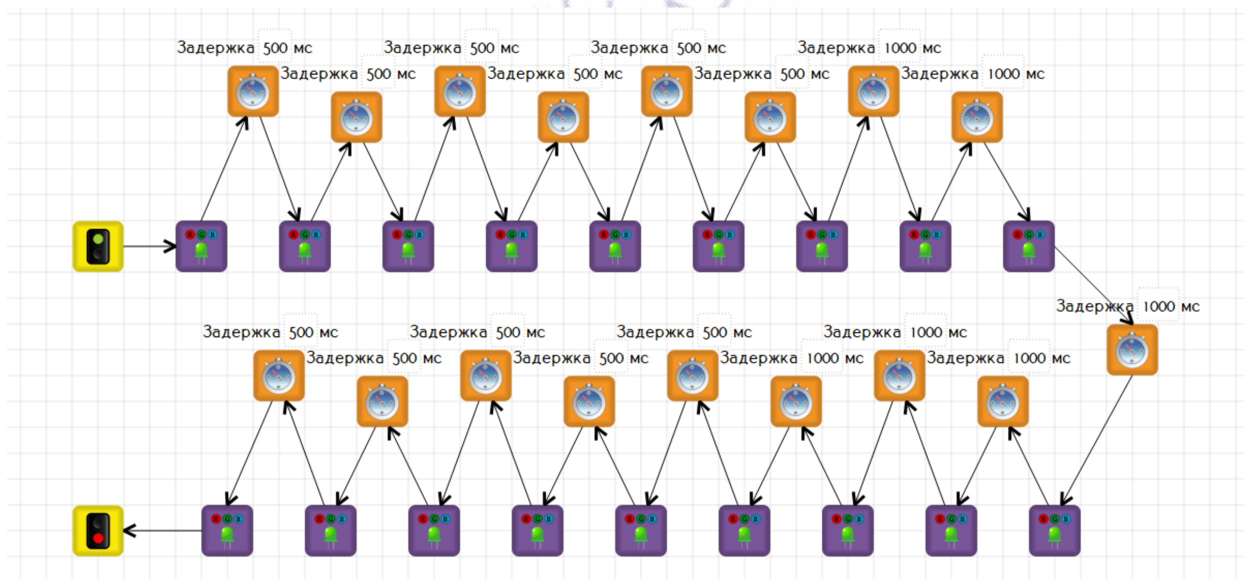
1. «Попавшие в беду»

Одной из самых известных и важных последовательностей на азбуке Морзе является международный сигнал бедствия – SOS.

Он применяется повсеместно приблизительно с начала 20 века и является последовательностью из 9 знаков: 3 точки, 3 тире, 3 точки.

Попробуем сделать так, чтобы коптер подавал сигнал бедствия своими светодиодами.

В TRIK Studio создадим цепочку из 18 светодиодов и 16 таймеров между ними, так как для каждого включения светодиода нам необходим блок, выключающий его.

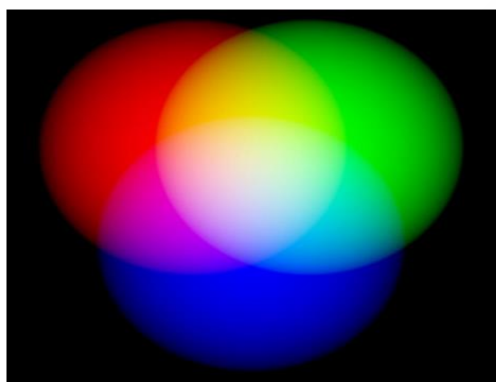


В данном примере цвет и задержку выбирайте сами. Однако, так как точки должны мигать быстрее, чем тире, между точками проставим таймер по 500 мс, а между тире 1000 мс.

Занятие 4. Программирование на языке Lua – управление светодиодами.

Теория:

RGB-светодиод – это три одноцветных кристалла совмещенные в одном корпусе. Название RGB расшифровывается, как Red – красный, Green – зеленый, Blue – синий соответственно цветам, которые излучает каждый из кристаллов.



Как вы знаете, путём смешивания трёх основных цветов можно получить любой другой цвет.

Также каждый светодиод может светить с разной яркостью.

Упрощённо говоря, яркость света – это отношение силы света, которую испускает поверхность к площади этой поверхности.

По тем же механизмам работает и наш глаз. В нем есть палочки, ответственные за восприятие яркости, и колбочки, которые воспринимают цвет. Этих колбочек в глазу три типа, по основным цветам – красные, синие и зеленые, каждая из которых реагирует на свой.

Практика:

1. Самостоятельное составление таблицы цветов.

ФОНД СОДЕЙСТВИЯ РАЗВИТИЮ ВОЕННОГО ОБРАЗОВАНИЯ

В Пионере любой цвет на светодиоде кодируется тремя параметрами от 0 до 1, где 0 это отсутствие свечения в данном цвете, а 1 — это самый яркий цвет.

Попробуйте сами составить таблицу цветов радуги, подставляя различные цифры в блок светодиода.

Таблицы в Lua создаются по одинаковой схеме:

local <имя вашей таблицы> = {
-- содержимое таблицы
}

, где вместо «имя таблицы» указывайте любое, я поставлю colors.

```
local colors = {
-- содержимое вашей таблицы
}
```

Содержимое ячеек таблицы может быть, как именованное, так и неименованное. Сами ячейки могут содержать в себе один параметр, другие таблицы или целые массивы информации.

Теперь, каждый цвет закодируем в формате {красный, зеленый, синий}, допустим, для красного цвета:

```
local colors = {
    {1,0,0} -- {r, g, b}
}
```

Создадим таблицу всех цветов радуги

ФОНД СОДЕЙСТВИЯ РАЗВИТИЮ ВОЕННОГО ОБРАЗОВАНИЯ

```
--таблица цветов радуги
local colors = {
    {1,0,0},       -- красный
    {1,0.15,0},    -- оранжевый
    {1,1,0},       -- желтый
    {0,1,0},       -- зелёный
    {0,1,1},       -- голубой
    {0,0,1},       -- синий
    {1,0,1}        -- фиолетовый
}
```

2. Код Lua для управления светодиодами.

Откроем Pioneer Station и создадим новый файл.

Для работы со светодиодами необходимо их инициализировать. Для этого введем переменную, в которую пропишем число светодиодов и команду *Ledbar.new*, которая их инициализирует.

```
1  -- создание порта управления светодиодом
2  local ledNumber = 4
3  local ledbar = Ledbar.new(ledNumber)
```

Далее необходимо прописать функцию callback, в которую должны приходит различные состояния при полёте, но, так как в этом задании не будет полета, оставим функцию пустой.

```
5  -- функция обработки событий, автоматически вызывается автопилотом
6  function callback(event)
7  end
```

После этого мы можем прописать код, который будет управлять светодиодами.

Каждый блок прописывается одной командой – Светодиод – *ledbar:set()*, Таймер – sleep().

```
 9  ledbar:set(0,1,0,0)
10
11  sleep(1)
```

В *ledbar:set()* в скобки пропишем 4 параметра: номер светодиода и значения красного, зеленого и синих цветов соответственно. На скриншоте загорится светодиод номер 0 красным цветом.

Команда *sleep()* требует только один параметр – время задержки в секундах.

Теперь мы можем сделать так, чтобы светодиод загорался всеми цветами радуги, используя таблицу цветов.

```
 9  ledbar:set(0,1,0,0)
10  sleep(1)
11  ledbar:set(0,0,1,0)
12  sleep(1)
13  ledbar:set(0,0,0,1)
14  sleep(1)
15
```

Данный пример переключает цвета: красный, зеленый и синий с задержкой в 1 секунду.

Также попробуйте поставить меньшую яркость, вместо 1 ставя 0.5.

Теперь воспользуемся таблицей, сделанной ранее.

Для начала напишем небольшую команду, которую будем использовать для удобства (она нужна для переноса данных из таблицы цветов в *ledbar:set*().

```
2  local ledNumber = 4
3  local ledbar = Ledbar.new(ledNumber)
4  local unpack = table.unpack()
```

local unpack = table.unpack

Воспользуемся ею при включении светодиода

```
16  ledbar:set(0, colors[1])
```

Первый параметр это номер светодиода, а цифра в квадратных скобках это порядковый номер цвета в нашей таблице, начиная с 1. Т.е. в данном примере светодиод загорится красным цветом.

ФОНД СОДЕЙСТВИЯ РАЗВИТИЮ ВОЕННОГО ОБРАЗОВАНИЯ

Занятие 5. Автономная навигация БАС – Оптическое позиционирование

Теория:

Автономный полет Пионер производит с помощью дополнительных модулей. Варианта позиционирования два: оптическое и ультразвуковое (OPT и LPS соответственно). На расширительной плате (или в виде доп. модуля) находится камера OPT. Это камера, которая работает на высоте до 2 м.

Камера оптического позиционировани смотрит строго вниз и по изменению картинки (и данных о высоте, полученных также с этой камеры) рассчитывает изменение координаты коптера.

Использование OPT удобно тем, что не требует дополнительного оборудования. Однако точности с помощью камеры оптического позиционирования не достичь.

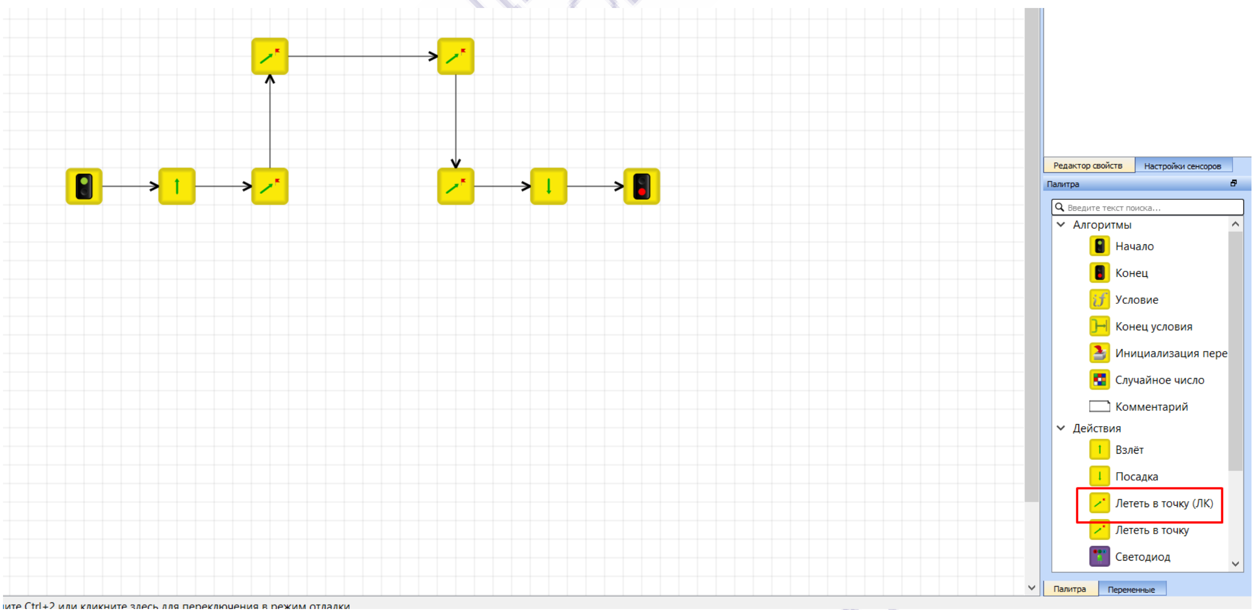
Одним из факторов является то, что очень часто на полу используется симметричный рисунок. Также она сильно зависит от освещенности покрытия. При резких поворотах коптера картинка может немного съезжать, вызыва смещение коптера.

Практика:

1. Полет по квадрату с помощью OPT.

В TRIK Studio создадим небольшой блок, в котором будет взлет, перемещение по 4 точкам и посадка.

ФОНД СОДЕЙСТВИЯ РАЗВИТИЮ ВОЕННОГО ОБРАЗОВАНИЯ



Выбираем блоки «Лететь в точку (ЛК)», ЛК – локальные координаты.

В этой системе координат место взлета Пионера будет иметь координаты (0,0). Проставьте точки для квадрата с гранью 1 м и на удобной высоте.

Самостоятельно проставьте код для включения любого светодиода меняющимся цветом после каждой точки.

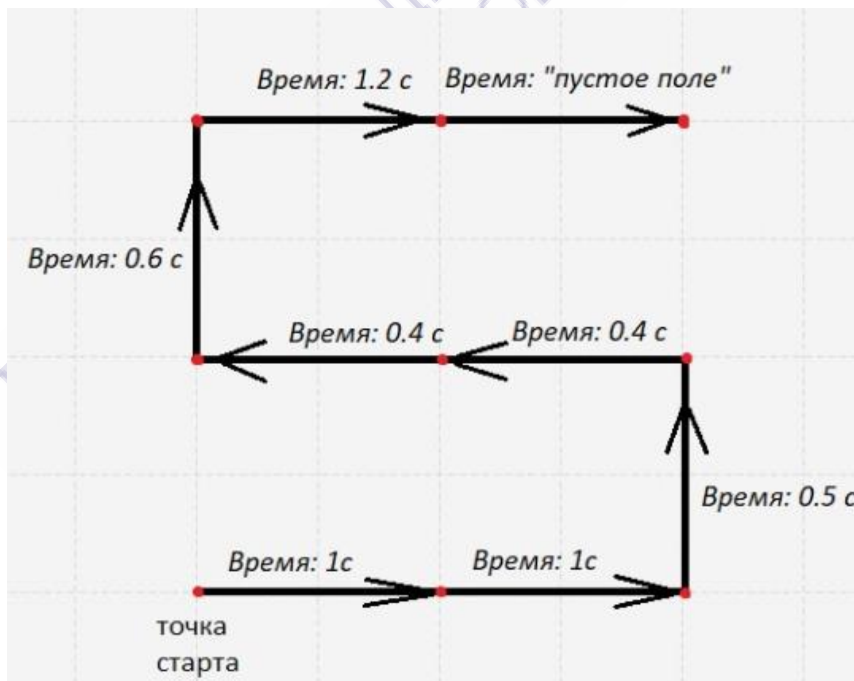
Основная функция для перемещения в системе локальных координат в Lua:

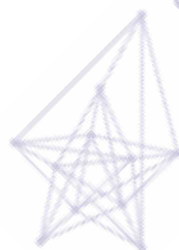
ap.goToLocalPoint(x,y,z)

2. Полет по змейке.

Самостоятельно пропишите код, с помощью которого квадрокоптер будет лететь по змейке, допустим, как на картинке.

ФОНД СОДЕЙСТВИЯ РАЗВИТИЮ ВОЕННОГО ОБРАЗОВАНИЯ


Время: 1.2 с
Время: "пустое поле"
Время: 0.6 с
Время: 0.4 с
Время: 0.4 с
Время: 0.5 с
Время: 1с
Время: 1с
точка
старта



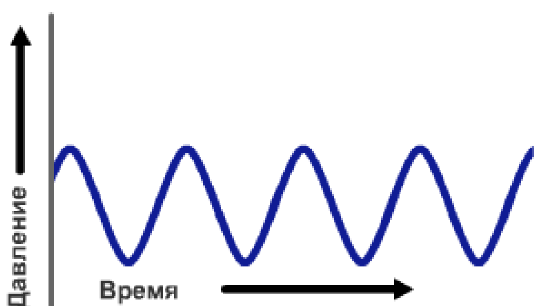
Занятие 6. Автономная навигация БАС – Маяки.

Теория:

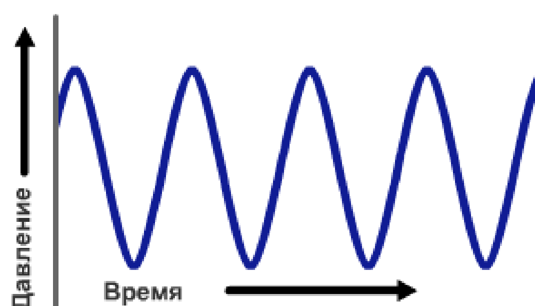
Звук – явления передачи колебаний в разных средах в виде упругих волн.

Так как это волна, звук характеризуется амплитудой и частотой.

Амплитуда влияет на громкость звука.

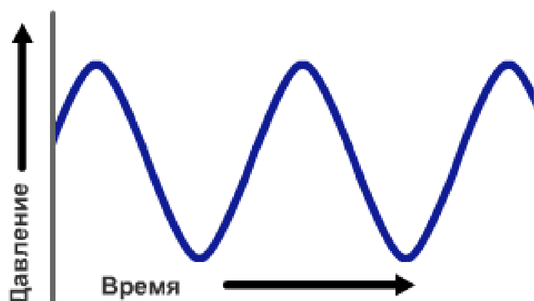


ТИХО

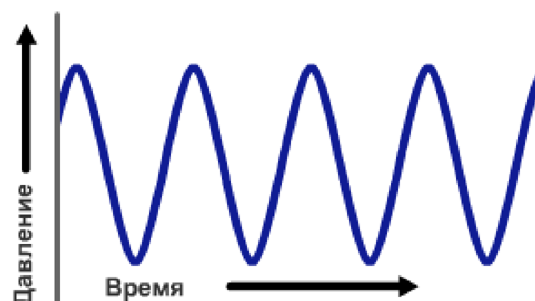


ГРОМКО

Частота влияет на высоту (тон) звука.



НИЗКИЙ



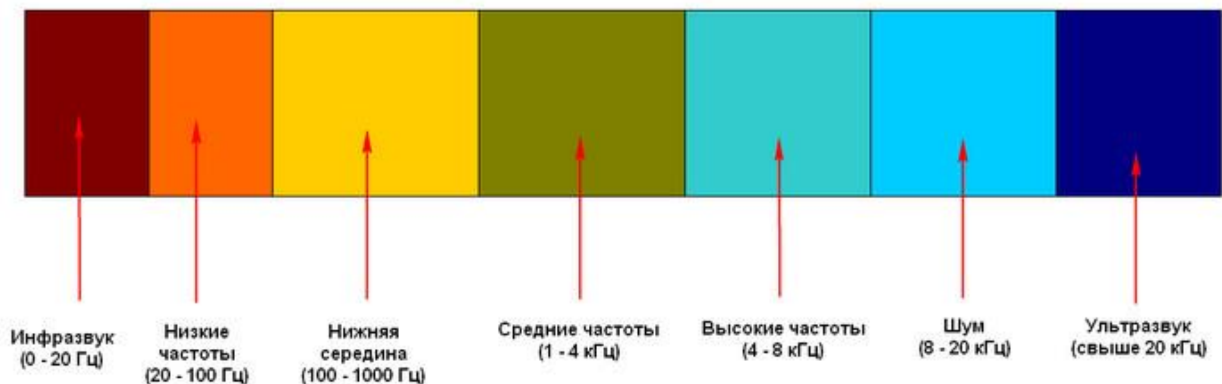
ВЫСОКИЙ

Человеческое ухо воспринимает звук от 16 Гц до 20000 Гц.

Герц (Гц) – число колебаний в секунду.

В зависимости от частоты, звук разделяют на несколько групп.

ФОНД СОДЕЙСТВИЯ РАЗВИТИЮ ВОЕННОГО ОБРАЗОВАНИЯ



Свыше 1 ГГц – гиперзвук, который имеет некоторые особенности (не распространяется при н.у. в воздухе, взаимодействует с фотонами).

Ультразвук – звуковые волны, имеющие частоту от 20 кГц до 1 ГГц, и не воспринимаемые человеческим ухом.

Они имеют большую энергию (из-за частоты), вызывают деформацию среды (допустим, могут разорвать животную клетку, что используется в биологии). Из-за высокой энергии ультразвуковые волны отражаются от многих поверхностей.

Этим явлением пользуются многие животные (эхолокация), такие как летучие мыши, дельфины, некоторые бабочки и птицы.

Система навигации в помещении «Локус» также основана на ультразвуке. Она состоит из четырех УЗ излучателей (для корректной работы требуются только 3, четвертый необходим для избыточности сигнала в случае помех с одного датчика), каждый из которых распространяет УЗ волны своей частоты.

На Пионер сверху крепится бортовой модуль навигации в помещении, который представляет собой плату с двум микрофонами, которые принимают сигнал с излучателей, и, по разнице в фазе звука и времени получения сигнала, рассчитывают координату с точностью до пары сантиметров.

<u>Практика:</u>

1. Настройка системы навигации Локус.

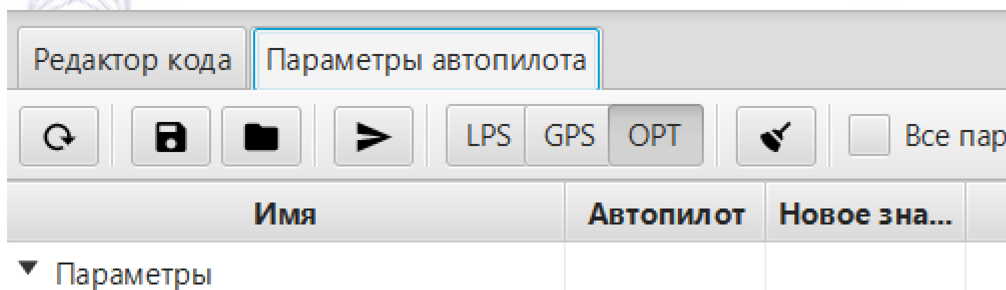
ФОНД СОДЕЙСТВИЯ РАЗВИТИЮ ВОЕННОГО ОБРАЗОВАНИЯ

На сайте есть подробный pdf файл, где рассказывается подробно об установке системы.

Самое важное, чтобы излучатели были закреплены максимально жестко, иначе координаты будут сдвигаться. Также желательно покрытие, которое будет минимально отражать звуковой сигнал.

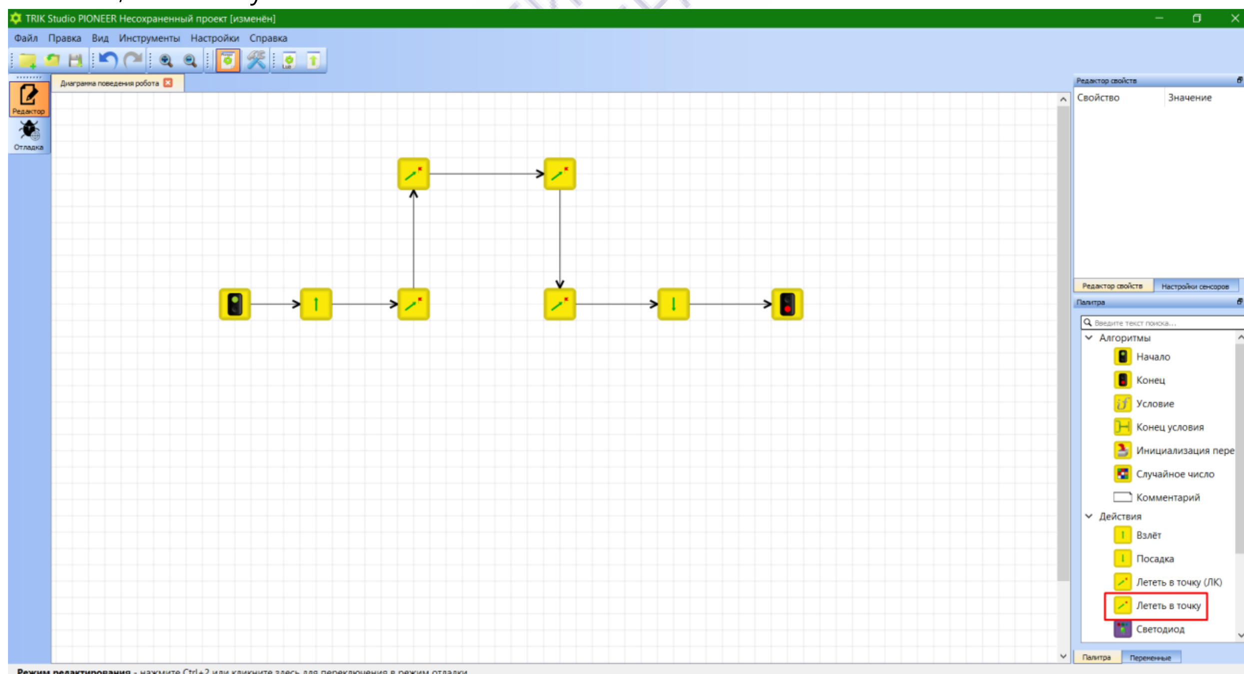
Минимальная высота для установки излучателей – 2 м, расстояние между излучателями – минимум 3 м.

Для работы с локусом необходимо скачать приложение, которое есть на сайте.



В Pioneer Station в параметрах автопилота переключите OPT на LPS

После настройки системы предлагаю вам прописать код из предыдущего занятия, использу вместо OPT – LPS.



Так выглядит в TRIK Studio полет по квадрату. Координаты точек квадрата проставьте в зависимости от координат в вашем локусе.

Самостоятельно попробуйте сделать змейку из предыдущего урока и треугольник в вертикальной плоскости.

Пособие и примеры по ссылке https://github.com/Slond/pioneer_edu

Занятие 7. Автономная навигация БАС – Точность

Теория:

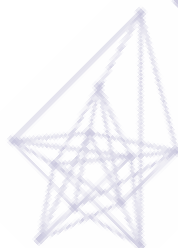
LPS, в отличие от OPT не зависит от освещения. Эта система существенно точнее. Из минусов, Локус очень зависит от устойчивого положения излучателей и отсутствии препятствий между Пионером и излучателями.

В системе LPS одновременно можно запустить несколько дронов, главное, чтобы один не перекрывал сигнал для другого.

Практика:

На данном занятии необходимо пролететь пионером какую-либо фигуру и приземлиться в круг диаметром 5-10 см.

Как пример, нарисовать «елочку» в вертикальной плоскости и приземлиться.



ФОНД СОДЕЙСТВИЯ РАЗВИТИЮ ВОЕННОГО ОБРАЗОВАНИЯ

Занятие 8. Объединение пилотируемого полета и автономной программы

Теория:

Настройка пульта для работы с квадрокоптером хорошо прописана в инструкции по сборке Пионера, начиная со стр.24.

Каждый тумблер на пульте имеет свой канал связи с Пионером. Всего их 8: 1-4 отвечают за стики управления (по осям x и y соответственно), 5-8 за тумблеры SwA, SwB, SwC и SwD в соответствии, указанном в инструкции (Настройки -> “Aux. channels”), а именно – 5 – SwC, 6 – SwD, 7 – SwB, 8 – SwA.

Каналы используются для смены каких-либо состояний в полёте. Например, тумблер SwB ставится в крайнее нижнее положение для перехода в режим автономного пилотирования, среднее положение тумблера SwC используется для зависания дрона на одной высоте.

Практика:

1. Модуль магнитного захвата груза.

Магнит крепится снизу на коптер в соответствии с длиной паза, так как один паз длиннее, модуль всегда встает в правильном положении. При необходимости можно поставить стойки и вкрутить модуль винтами.

На пульте необходимо настроить 8 канал на канал SwA (Настройки -> “Aux. channels” -> “SwX” -> “SwA”)

После этого приступим к написанию программы. Для начала инициализируем светодиоды (теперь их стало 8, 4 на плате и 4 на магните), магнит и пропишем функцию callback.

```
1  local magneto = Gpio.new(Gpio.C, 3, Gpio.OUTPUT) -- инициализируем управление модулем груза
2  local led_number = 8 -- задаем количество светодиодов (4 на базовой плате и еще 4 на модуле груза)
3  -- инициализируем светодиоды
4  local leds = Ledbar.new(led_number)
5  local rc = Sensors.rc
6  function callback(event)
7  end
```

Далее напишем функцию, которая будет менять цвет всех светодиодов.

Синтаксис функции такой:

local function <имя функции>()

<Блок тела функции>

end

Внутри функции введем цикл for. Этот цикл с каким-то шагом выполняет то, что у него внутри (по умолчанию шаг = 1)

for <начало>, <конец>, [<шаг>] do

<Блок тела цикла>

end

Квадратные скобки у <шаг> означают то, что этот параметр необязателен.

```
10  -- функция смены цвета светодиодов
11  local function changeColor(red, green, blue)
12      for i=0, led_number - 1, 1 do
13          leds:set(i, red, green, blue)
14      end
15  end
```

Данная функция *changeColor* требует при вызове в скобках три параметра – цвет светодиода, а затем выполняет *leds:set()* для каждого i от 0 до led_number – 1, так как нумерация светодиодов начинается с 0.

Далее напишем таймер. Таймер выполняет какую-либо функцию с заданным интервалом. В нашем случае таймер должен каждые N секунд выполнять проверку положения тумблера, где N достаточно малое время, чтобы не было задержки.

```
17  cargoTimer = Timer.new(0.1, function () -- создаем таймер, который будет вызывать нашу функцию 10 раз в секунуду
18      _, _, _, _, _, _, _, ch8 = rc() -- считываем сигнал с 8 канала на пульте, значение от -1 до 1
```

На строке 18 описано получение данных именно с 8 канала.

После этого сделаем разветвление программы, условием которого будет положение тумблера.

Для этого используется блок if/else.

if <условие> then

<блок кода, при выполнении условия>

else

<блок кода, при не выполнении условия>

end

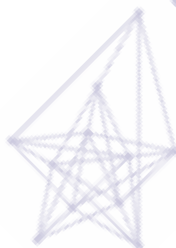
```
17 cargoTimer = Timer.new(0.1, function () -- создаем таймер, который будет вызывать нашу функцию 10 раз в секунуду
18     _, _, _, _, _, _, _, ch8 = rc() -- считываем сигнал с 8 канала на пульте, значение от -1 до 1
19     if(ch8 < 0) then  -- если сигнал с пульта -1 (SWA вверх), то включаем
20         magneto:set()
21         changeColor(0, 1, 0) -- и сигнализируем об активации зеленым цветом
22     else if(ch8 > 0) then -- если сигнал с пульта 1 (SWA вниз), то выключаем
23         magneto:reset()
24         changeColor(1, 0, 0) -- когда магнит отключен, светодиоды горят красным
25     else -- синий мигающий цвет светодиодов сигнализирует об отсутствии сигнала на восьмом канале
26         for i=1, 8 do
27             changeColor(0,0,1)
28             sleep(0.5)
29             changeColor(0,0,0)
30             sleep(0.5)
31         end
32     end
33 end
34 end)
35
```

Не запутайтесь в числе *end* в конце блока.

Остается последняя строка кода, которая включит таймер

```
 -- запускаем таймер
cargoTimer:start()
```

Попробуйте самостоятельно добавить sleep перед выключением магнита и посмотреть, что получится.



ФОНД СОДЕЙСТВИЯ РАЗВИТИЮ ВОЕННОГО ОБРАЗОВАНИЯ

Занятие 9. Работа с автопилотом

Теория:

Автопилот – незаменимая часть в Пионере. Полётная плата постоянно получает какую-либо информацию. Наша основная цель – научить коптер реагировать на неё автономно.

Все данные приходят из двух источников: пульт ДУ и встроенные датчики. Для получения данных с пульта необходим радиоприемник, но об этом подробнее поговорим на следующем занятии.

Список основных показаний, которые может получить плата с датчиков есть в документации по Lua, здесь мы приведем некоторые из них.

1. Положение, скорость и угол поворота в системе LPS.
2. Значения высот с разных датчиков.
3. Данные с пульта ДУ

Все эти данные объединяются в класс Sensors, из которого их можно получить.

Помимо этого, автопилот ответственный за функцию callback(event), в которой мы можем прописывать различные ответы на события (список всех событий также есть в документации).

Практика:

1. Получение данных о высоте.

Пионер получает информацию о высоте с разных датчиков, в зависимости от значения высоты.

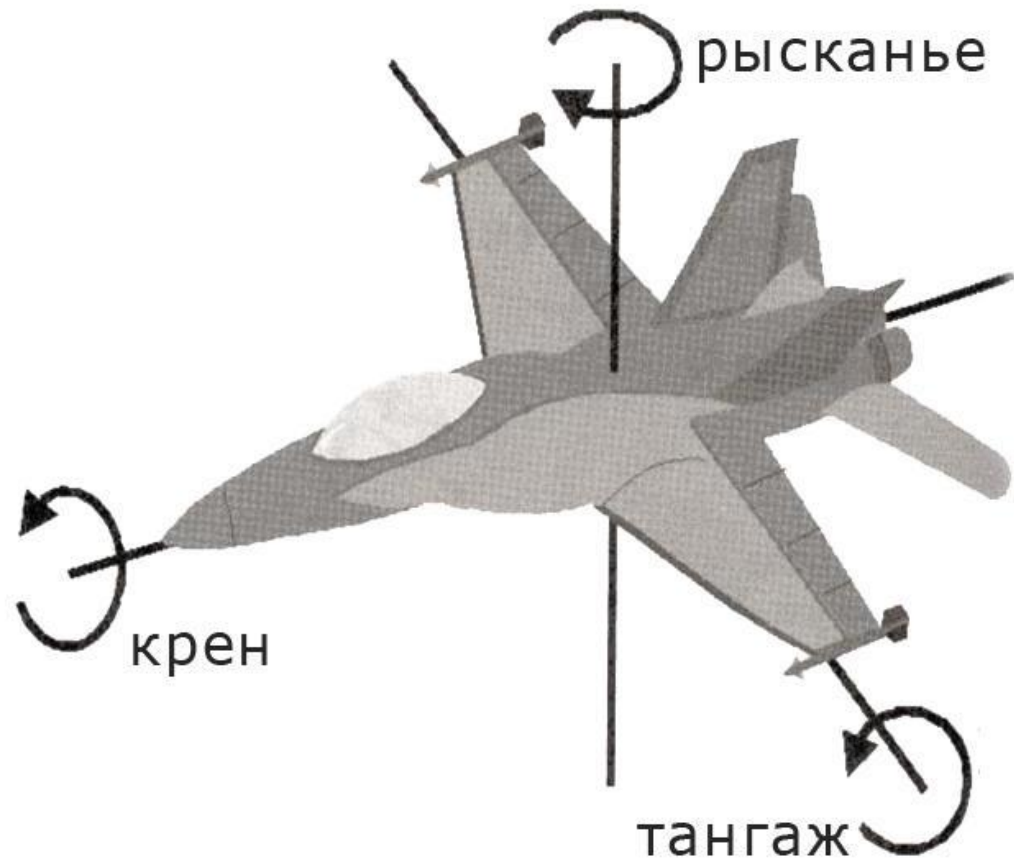
На высотах до 3 метров основной источник информации – дальномер, расположенный на камере оптического зрения. С него приходят данные о

ФОНД СОДЕЙСТВИЯ РАЗВИТИЮ ВОЕННОГО ОБРАЗОВАНИЯ

Занятие 9. Программирование в TRIK Studio – Рыскание.

Теория:

Любой летательный аппарат может поворачиваться в 3 плоскостях. В зависимости от плоскости каждый поворот называется по-разному.



Тангаж – поворот по поперечной оси, т.е. нос летательного аппарата вверх/вниз.

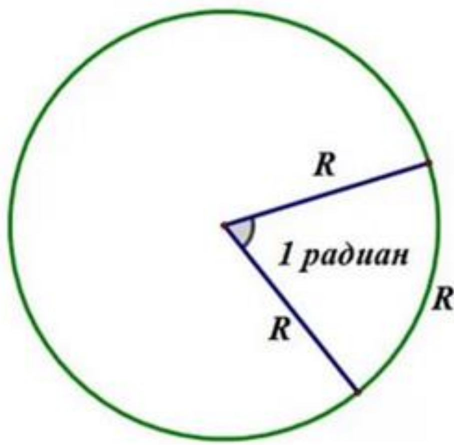
Крен – поворот по продольной оси, при нём крылья будут подниматься вверх/вниз в зависимости от направления.

Рыскание – поворот по вертикальной оси, нос летательного аппарата будет двигаться влево/вправо.

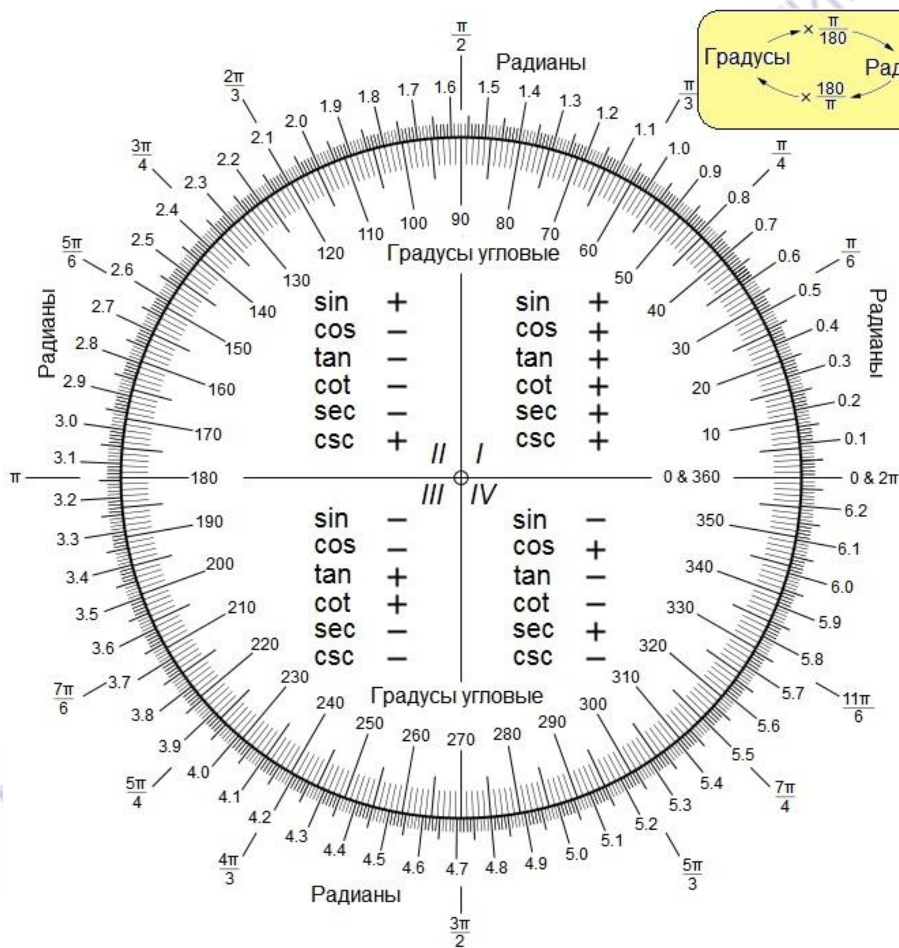
Из всех видов поворота нам важно именно рыскание.

Рыскание проводится на угол, однако в программировании важны радианы.

Радиан – это угол, который образуется при наложении радиуса на дугу окружности.

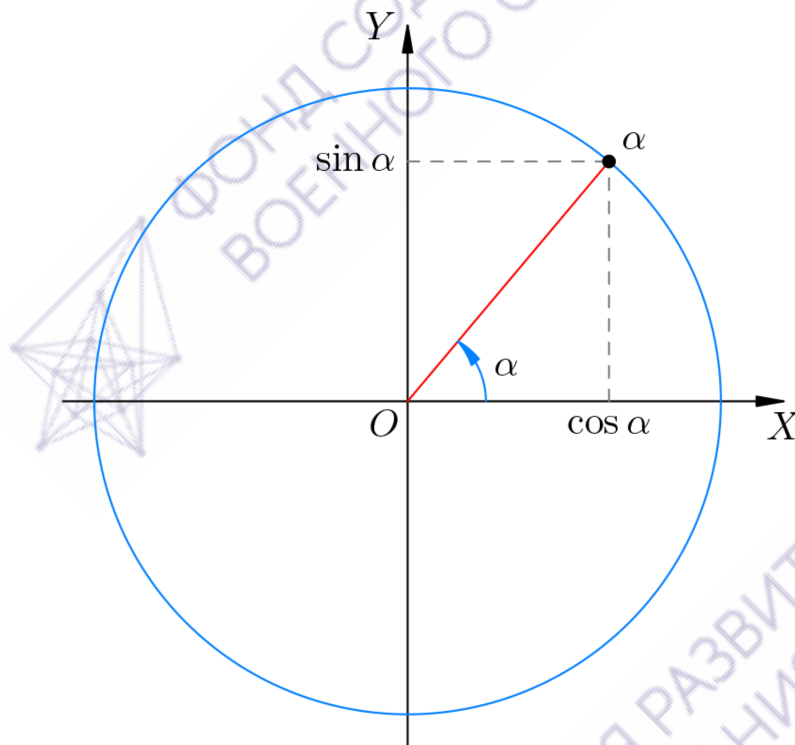


Радианы нам важны для понимания тригонометрического круга.



Так как часто мы будем запускать дрона по окружности, нам нужно рассчитать точки, к которым он будет лететь с помощью *ap.goToLocalPoint(x,y,z)*.

Для вычисления координаты любой точки на окружности, зная лишь угол, нам необходимы синус и косинус (sin и cos).



Все значения синуса и косинуса любого угла есть в тригонометрических таблицах (как пример, таблица Брадиса) или в калькуляторе/интернете.

<u>Практика:</u>

1. Полет по окружности с ориентацией по курсу.

Для полёта по любой траектории нужно получить координату и угол поворота.

Для примера мы рассчитаем 8 точек (каждые 45 градусов).

Для расчета координаты воспользуемся формулой x = R * cosA, y = R * sinA.

R примем за 1 метр.

Вот полученные координаты в виде таблицы.

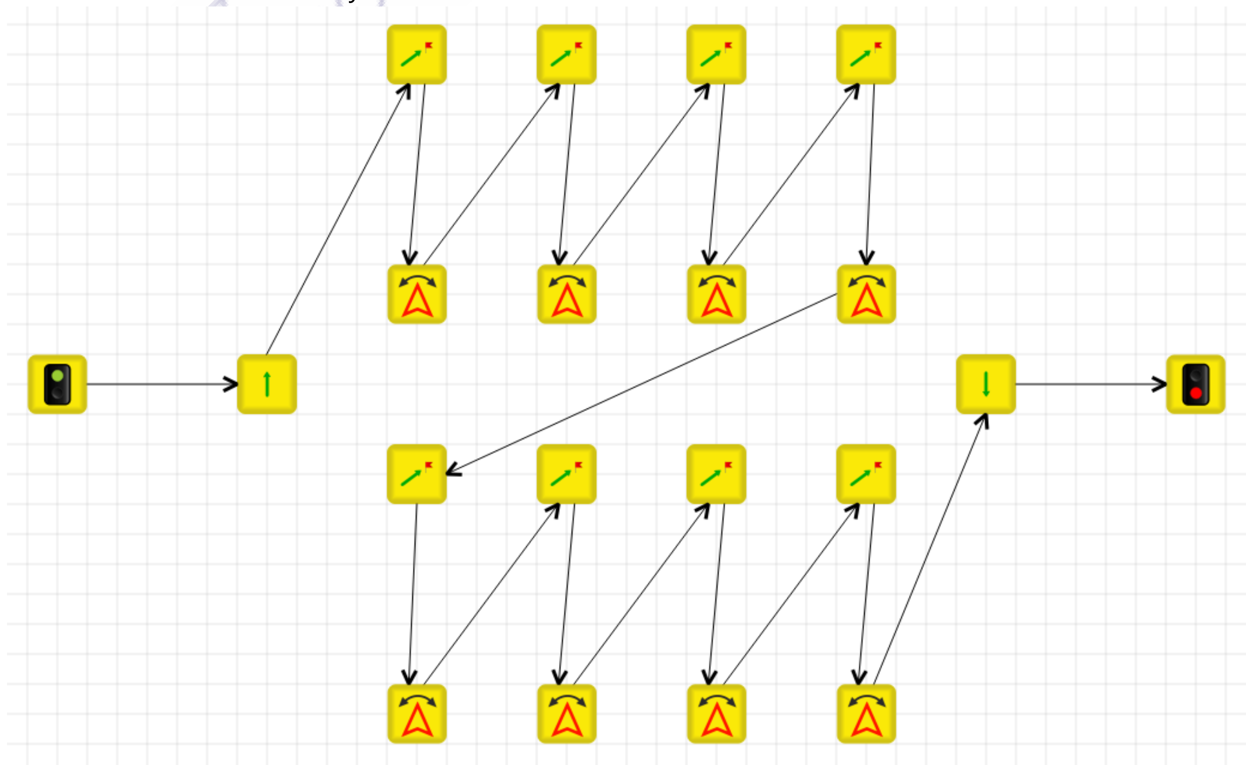
№	1	2	3	4	5	6	7	8
x	1	0,7	0	-0,7	-1	-0,7	0	0,7
y	0	0,7	1	0,7	0	-0,7	-1	-0,7

ФОНД СОДЕЙСТВИЯ РАЗВИТИЮ ВОЕННОГО ОБРАЗОВАНИЯ

Угол тангажа будет -45, так как точки поставлены против часовой стрелки.

Высоту Z поставим 1 метр.

В TRIK Studio получается такая схема:



ФОНД СОДЕЙСТВИЯ РАЗВИТИЮ ВОЕННОГО ОБРАЗОВАНИЯ

Пособие и примеры по ссылке https://github.com/Slond/pioneer_edu

Занятие 10. Соревнование «Скорая помощь»

Цель занятия – в соревновательной форме проверить, насколько ученики хорошо освоили работу в LPS/OPT и работу со светодиодами.

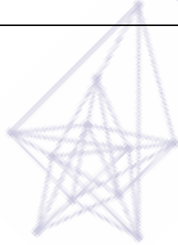
Для занятия необходимо построить полосу препятствий из доступных материалов. Целью соревнований является успешный полет по полосе с минимумом нарушений (касание объектов, светодиодная индикация).

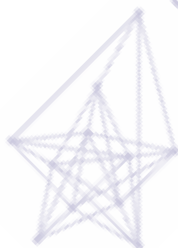
Таблицу оценивания предлагаем составить самостоятельно, вот пример:

Подготовка к полёту, взлёт	1. Включение питания квадрокоптера в безопасной воздушной зоне 2. Включение двигателей в безопасной воздушной зоне 3. Выполнение взлета в стартовой зоне 4. Полёт без касания пола 5. Полёт без касания сетки 6. Полёт без касания препятствий(если есть)
Выполнение полёта в горизонтальной плоскости	1. Полёт по траектории "квадрат" по часовой стрелке (квадрокоптер направлен вперед) 2. Полёт по траектории "квадрат" против часовой стрелки (квадрокоптер направлен вперед) 3. Полёт по траектории "квадрат" по часовой стрелке (квадрокоптер направлен на оператора) 4. Полёт по траектории "квадрат" против часовой стрелки (квадрокоптер направлен на оператора)
Выполнение полёта в горизонтальной плоскости с рысканьем	1. Полёт по траектории "квадрат" по часовой стрелке (квадрокоптер направлен вперед) 2. Полёт по траектории "квадрат" против часовой стрелки (квадрокоптер направлен вперед) 3. Полёт по траектории "квадрат" по часовой стрелке (квадрокоптер направлен на оператора) 4. Полёт по траектории "квадрат" против часовой стрелки (квадрокоптер направлен на оператора) 5. Полёт по траектории "квадрат" по часовой стрелке (квадрокоптер поворачивается по направлению движения) 6. Полёт по траектории "квадрат" против часовой стрелки (квадрокоптер поворачивается по направлению движения)

ФОНД СОДЕЙСТВИЯ РАЗВИТИЮ ВОЕННОГО ОБРАЗОВАНИЯ

Выполнение посадки, выключение БВС	1. Выполнение посадки в стартовой зоне 2. Выключение двигателей квадрокоптера в безопасном воздушном пространстве 3. Выключение питания квадрокоптера в безопасном воздушном пространстве





ФОНД СОДЕЙСТВИЯ РАЗВИТИЮ ВОЕННОГО ОБРАЗОВАНИЯ

Занятие 11. Программирование на языке Lua – Таблицы

Теория:

Если вы имеете опыт программирования, то таблицы в Lua это аналог массива в любом другом языке.

Если же нет, таблица – это некое хранилище разных данных. В таблицах можно хранить любой тип данных, даже другие таблицы.

С таблицами мы уже сталкивались при создании таблицы цветов, но вспомним синтаксис.

local <название таблицы> = {

<Данные внутри таблицы>

}

Повторите основы тригонометрии из предыдущего занятия.

Практика:

1. Полет по массиву точек.

Сперва по аналогии с предыдущими занятиями пропишем код для инициализации светодиодов, функцию смены цвета на всех светодиодах и таблицу цветов.

ФОНД СОДЕЙСТВИЯ РАЗВИТИЮ ВОЕННОГО ОБРАЗОВАНИЯ

```
3   -- Упрощение вызова функции распаковки таблиц из модуля table
4   local unpack = table.unpack
5
6   -- Количество светодиодов на базовой плате
7   local ledNumber = 4
8   -- Создание порта управления светодиодами
9   local leds = Ledbar.new(ledNumber)
10
11  -- Функция смены цвета светодиодов
12  local function changeColor(color)
13      -- Поочередное изменение цвета каждого из 4-х светодиодов
14      for i=0, ledNumber - 1, 1 do
15          leds:set(i, unpack(color))
16      end
17  end
18
19  -- Таблица цветов в формате RGB для передачи в функцию changeColor
20  local colors = {
21          {1, 0, 0}, -- красный
22          {0, 1, 0}, -- зеленый
23          {0, 0, 1}, -- синий
24          {1, 1, 0}, -- желтый
25          {1, 0, 1}, -- фиолетовый
26          {0, 1, 1}, -- бирюзовый
27          {1, 1, 1}, -- белый
28          {0, 0, 0}  -- черный/отключение светодиодов
29  }
```

Далее создадим таблицу с точками полёта, для примера это будет правильная звезда. Координаты точек указаны на скриншоте (попробуйте их высчитать самостоятельно).

```
31  -- Таблица точек полетного задания в формате {x,y,z}
32  local points = {
33          {0, 0, 0.7},
34          {0.25, 0.77, 0.7},
35          {0.5, 0, 0.7},
36          {-0.15, 0.48, 0.7},
37          {0.65, 0.48, 0.7}
38  }
```

В функции callback необходимо прописать, что будет делать коптер при различных событиях. Мы будем отлавливать четыре из них:

1. Коптер успешно взлетел
2. Коптер добрался до нужной точки
3. Коптер ударился о препятствие
4. Коптер приземлился

```
67 -- Функция обработки событий, автоматически вызывается автопилотом
68 function callback(event)
69     -- Когда коптер поднялся на высоту взлета Flight_com_takeoffAlt, переходим к полету по точкам
70     if(event == Ev.TAKEOFF_COMPLETE) then
71         nextPoint()
72     end
73     -- Когда коптер достиг текущей точки, переходим к следующей
74     if(event == Ev.POINT_REACHED) then
75         nextPoint()
76     end
77     -- Когда коптер приземлился, выключаем светодиоды
78     if (event == Ev.COPTER_LANDED) then
79         changeColor(colors[8])
80     end
81     -- При столкновении отключить двигатели
82     if (event == Ev.SHOCK) then
83         ap.push(Ev.ENGINES_DISARM)
84 end
```

Далее создадим таблицу с действиями, которые должен выполнить Пионер. Назовем ее “actions”. Пример этой таблицы вы можете посмотреть в коде Lua предыдущего занятия.

В этой таблице необходимо прописать различные события, которые будут вызываться. В конце каждого события нужно прописать, какое действие запустить.

Как пример приведем список дел каждое утро:

actions = {

if <проснулся> then

иди умываться

след.действие = “умывание”

end,

if <умывание> then

...

И так далее.

Список действий для полета по точкам у дрона следующий:

1. Включить двигатели, после небольшой паузы взлететь
2. Добраться до точек 1-5
3. Приземлиться

Так как у Пионера нет индикации, кроме светодиодов, на каждой из точек будем зажигать их новым цветом, что будет говорить, на каком пункте списка actions сейчас дрон.

ФОНД СОДЕЙСТВИЯ РАЗВИТИЮ ВОЕННОГО ОБРАЗОВАНИЯ

После запуска двигателей необходимо подождать какое-то время, чтобы они включились на полную мощность, поэтому на взлете будем использовать таймеры.

NB! Таймеры на Пионере выполняются одновременно, что позволяет прописывать точное время запуска различных команд. В один момент может работать не более 15 таймеров. Команду sleep нежелательно использовать, так как она приостанавливает действие всего скрипта, включая таймеры.

```
41  -- таблица функций, вызываемых в зависимости от состояния
42  action = {
43      ["PREPARE_FLIGHT"] = function()
44          changeColor(colors[2]) -- смена цвета светодиодов на белый
45          Timer.callLater(2, function () ap.push(Ev.MCE_PREFLIGHT) end) -- через 2 секунды отправляем команду автопилоту на запуск моторов
46          Timer.callLater(4, function () changeColor(colors[3]) end)-- еще через 2 секунды (суммарно 4 секунды, так как таймеры запускаются сразу же друг за другом)
47          Timer.callLater(6, function ()
48              ap.push(Ev.MCE_TAKEOFF) -- еще через 2 секунды (суммарно через 6 секунд) отправляем команду автопилоту на взлет
49              curr_state = "FLIGHT_TO_FIRST_POINT" -- переход в следующее состояние
50          end)
51      end,
52      ["FLIGHT_TO_FIRST_POINT"] = function ()
53          changeColor(colors[4]) -- смена цвета светодиодов на желтый
54          Timer.callLater(2, function ()
55              ap.goToLocalPoint(unpack(points[1])) -- отправка команды автопилоту на полет к точке из списка points под номером 1
56              curr_state = "FLIGHT_TO_SECOND_POINT" -- переход в следующее состояние
57          end)
58      end,
59      ["FLIGHT_TO_SECOND_POINT"] = function ()
60          changeColor(colors[3]) -- смена цвета светодиодов на зеленый
61          Timer.callLater(2, function ()
62              ap.goToLocalPoint(unpack(points[2])) -- отправка команды автопилоту на полет к точке из списка points под номером 2
63              curr_state = "FLIGHT_TO_THIRD_POINT" -- переход в следующее состояние
64          end)
65      end,
```

Каждое состояние в таблице имеет общий вид:

[<Название функции>] = function()

<блок действий функции>

end

При взлете запускаются сразу три таймера: первый запускает двигатели, второй меняет цвет светодиодов и третий поднимает Пионер. Каждое действие выполняется спустя 2 секунды. Далее программа переходит на следующее состояние.

Полет по точкам – это 5 одинаковых блоков, которые отличаются только цветов светодиодов и точкой полета (по аналогии со светодиодами, прописываем *unpack(<имя списка>[<номер точки>]))*.

ФОНД СОДЕЙСТВИЯ РАЗВИТИЮ ВОЕННОГО ОБРАЗОВАНИЯ

```
    ["FLIGHT_TO_LAST_POINT"] = function (x)
        changeColor(colors[5]) -- смена цвета светодиодов на фиолетовый
        Timer.callLater(2, function ()
            ap.goToLocalPoint(unpack(points[1])) -- отправка команды автопилоту на полет к точке из списка points под номером 1
            curr_state = "PIONEER_LANDING" -- переход в следующее состояние
        end)
    end,
    ["PIONEER_LANDING"] = function ()
        changeColor(colors[6]) -- смена цвета светодиодов на синий
        Timer.callLater(2, function ()
            ap.push(Ev.MCE_LANDING) -- отправка команды автопилоту на посадку
        end)
    end
}
```

Последняя точка – это возврат в начало (точка 1), затем переход к посадке.

Остается в самом конце программы прописать два небольших блока: название первого состояния для Пионера и запуск всей программы.

```
-- переменная текущего состояния
local curr_state = "PREPARE_FLIGHT"
```

```
-- включаем светодиод (красный цвет)
changeColor(colors[1])
-- запускаем одноразовый таймер на 2 секунды, а когда он закончится, выполняем первую функцию из таблицы (подготовка к полету)
Timer.callLater(2, function () action[curr_state]() end)
```

2. Движение по окружности с помощью цикла.

Попробуем повторить движение по окружности из предыдущего занятия, но проставим большее количество точек. Суть программирования в облегчении работы, поэтому напишем сначала цикл, который создаст n точек, чем больше, тем лучше.

Для начала создадим пустой список для точек.

local points = {}

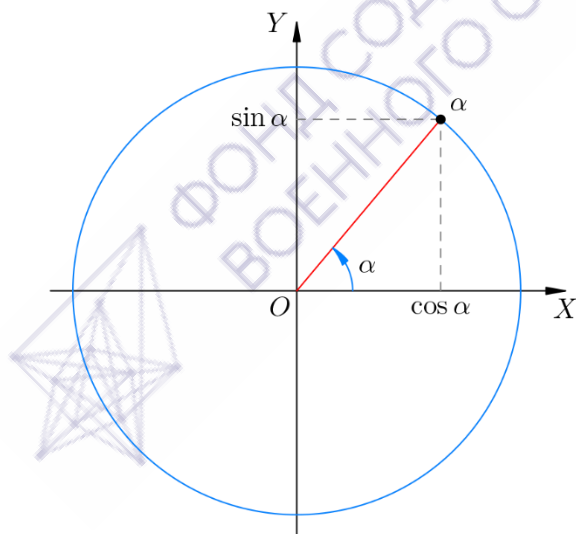
После этого используем цикл for.

for i=1, <количество точек>, do

points[i] = {<координаты точки>}

end

Для координат на окружности необходимо опять вернуться к тригонометрическому кругу.



Координата по x будет равна R окружности * cosA; по y, соответственно, R * sinA.

Получается, вид координаты в цикле будет {r*cosA, r*sinA, z}, где z – высота полета.

Радиус окружности пропишем в любом месте до цикла (в метрах)

local r = <значение>

Так как i в цикле увеличивается на 1, значение угла для каждой точки можно получить, как i * angle, заранее прописав значение этой переменной по аналогии с радиусом.

Функции синуса и косинуса в языке Lua прописываются из библиотеки math, и получаются вида math.sin() и math.cos(). Но эта библиотека в параметр требует значение угла в радианах, так что необходимо перевести угловое значение в дуговое – по формуле angle * math.pi / 180.

Таким образом, получается следующий цикл:

```
local r = 1
local angle = 30

for i=1, 12 do
    xCoord = r * math.cos(i * angle * math.pi / 180)
    yCoord = r * math.sin(i * angle * math.pi / 180)
    points[i] = {xCoord, yCoord, 0.7}
end
```

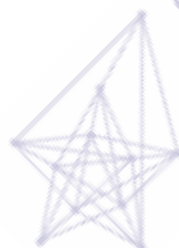
Также можно дописать внутри цикла код, который будет прекращать выполнение цикла после какого-либо условия, к примеру, когда угол уже больше 360 градусов.

ФОНД СОДЕЙСТВИЯ РАЗВИТИЮ ВОЕННОГО ОБРАЗОВАНИЯ

для этого после points[i] пропишем:

```
if i * angle > 360 then
    break
end
```

Как летать по этим точкам, не прописывая десятки действий рассмотрим на следующем занятии.



ФОНД СОДЕЙСТВИЯ РАЗВИТИЮ ВОЕННОГО ОБРАЗОВАНИЯ

Занятие 12. Программирование на языке Lua – Циклы.

Теория:

С циклами мы уже немного знакомы. На предыдущих занятиях использовался цикл for, это «цикл с счетчиком». Он выполняется определенное количество раз.

Кроме этого, мы будем пользоваться «безусловным» циклом. Это цикл, который выполняется при выполнении его условия, что может привести к образованию бесконечного цикла (если параметры в условии не меняются в цикле).

Синтаксис этого цикла такой:

while <условие цикла> do

<блок команд>

end

Чаще всего именно в блоке команд меняют переменную в условии.

Все циклы имеют несколько полезных операторов, break и continue. Первый досрочно завершает цикл, а второй начинает следующий проход, пропуская весь код внутри тела цикла.

```
while i<100 do
    if i == 50 then
        break
    end
    i = i + 1
end
```

Данный цикл прервется, когда счетчик дойдет до 50.

```
while i<100 do
    if i == 50 then
        continue
    end
    points[i] = {i, i, 0.7}
    i = i + 1
end
```

Данный цикл не создаст точку при i = 50, но все остальные создаст

Практика:

ФОНД СОДЕЙСТВИЯ РАЗВИТИЮ ВОЕННОГО ОБРАЗОВАНИЯ

1. Бесконечный полёт по окружности.

Для начала в Pioneer Station пропишем все команды, необходимые для инициализации светодиодов и упрощения работы.

```
-- количество светодиодов на основной плате пионера
local ledNumber = 4
-- создание порта управления светодиодами
local leds = Ledbar.new(ledNumber)
-- ассоциируем функцию распаковки таблиц из модуля table для упрощения
local unpack = table.unpack

-- переменная текущего состояния
local curr_state = "PREPARE_FLIGHT"



-- функция, изменяющая цвет 4-х RGB светодиодов на основной плате пионера
local function changeColor(color)
    -- проходим в цикле по всем светодиодам с 0 по 3
    for i=0, ledNumber - 1 do
        leds:set(i, unpack(color))
    end
end

-- таблица цветов в формате RGB для передачи в функцию changeColor
local colors = {
        {1, 0, 0}, -- красный
        {1, 1, 1}, -- белый
        {0, 1, 0}, -- зеленый
        {1, 1, 0}, -- желтый
        {1, 0, 1}, -- фиолетовый
        {0, 0, 1}, -- синий
        {0, 0, 0}  -- черный/отключение светодиодов
}

local r = 1
local angle = 30
local points = {}

for i=1, 12 do
    xCoord = r * math.cos(i * angle * math.pi / 180)
    yCoord = r * math.sin(i * angle * math.pi / 180)
    points[i] = {xCoord, yCoord, 0.7}
end
```

Также возьмем цикл, который создали на предыдущем занятии.

Теперь добавим список actions, в котором на этот раз будут только 3 состояния: взлет, полет к следующей точке и посадка.

Блоки взлета и посадки остаются прежними.

```
["PREPARE_FLIGHT"] = function()
    changeColor(colors[2]) -- смена цвета све
    Timer.callLater(2, function () ap.push(Ev
    Timer.callLater(4, function () changeColo
    Timer.callLater(6, function ()
        ap.push(Ev.MCE_TAKEOFF) -- еще через
        curr_state = "FLIGHT_TO_FIRST_POINT"
    end)
end,
```

```
["PIONEER_LANDING"] = function ()
    changeColor(colors[6]) -- смена цвета светодиодов на синий
    Timer.callLater(2, function ()
        ap.push(Ev.MCE_LANDING) -- отправка команды автопилоту на посадку
    end)
end
```

Все остальные блоки заменим на один, который назовем “FLIGHT” (не забудьте заменить название состояния в блоке взлета)

В теле состояния Flight пропишем безусловный цикл while, который будет выполняться, пока счетчик не превышает длину списка точек. Для этого в начале программы создадим переменную-счетчик j (local j = 1). Длину списка можно получить, написав #<название списка>.

Таким образом, инициализацией цикла будет while j < #points do

Внутри цикла пропишем перемещение к необходимой точке и увеличение счетчика.

```
["FLIGHT"] = function ()
    while j < #points do
        ap.goToLocalPoint(unpack(points[j]))
        j = j + 1
```

Однако, в Пионере можно поставить только одно перемещение в одном состоянии, иначе дрон полетит сразу к последней точке (что в нашем случае будет той же самой точкой, что и начало полета). Поэтому нам необходимо выйти из цикла, предварительно поставить выход из цикла с помощью команды break, предварительно зациклив состояние, поставив переход к нему самому (curr_state = “FLIGHT”)

```
["FLIGHT"] = function ()
    while j < #points do
        ap.goToLocalPoint(unpack(points[j]))
        j = j + 1
        curr_state = "FLIGHT"
        break
    end
```

И, необходимо как то обработать выход из цикла, в нашем случае Пионер должен перейти в состояние посадки.

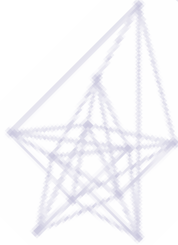
```
["FLIGHT"] = function ()
    while j < #points do
        ap.goToLocalPoint(unpack(points[j]))
        j = j + 1
        curr_state = "FLIGHT"
        break
    end
    curr_state = "PIONEER_LANDING"
end,
["PIONEER_LANDING"] = function ()
    changeColor(colors[6]) -- смена цвета свето
    Timer.callLater(2, function ()
        ap.push(Ev.MCE_LANDING) -- отправка ком
    end)
end
```

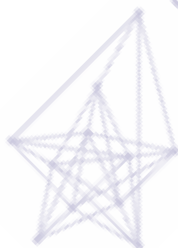
Таким образом, цикл будет выполняться до тех пор, пока не пройдет все точки, а после этого пойдет на посадку.

Осталось прописать функцию callback и начало программы, этот код возьмем из предыдущего занятия.

```
-- функция обработки событий, автоматически вызывается автопилотом
function callback(event)
    -- если достигнута необходимая высота, то выполняем функцию из таблицы, соответствующую текущему состоянию
    if (event == Ev.TAKEOFF_COMPLETE) then
        action[curr_state]()
    end
    -- если пионер с чем-то столкнулся, то зажигаем светодиоды красным и выключаем двигатели
    if (event == Ev.SHOCK) then
        changeColor(colors[1])
        ap.push(ENGINES_DISARM)
    end
    -- если пионер достигнул точки, то выполняем функцию из таблицы, соответствующую текущему состоянию
    if (event == Ev.POINT_REACHED) then
        action[curr_state]()
    end
    -- если пионер приземлился, то выключаем светодиоды
    if (event == Ev.COPTER_LANDED) then
        changeColor(colors[7])
    end
end


-- включаем светодиод (красный цвет)
changeColor(colors[1])
-- запускаем одноразовый таймер на 2 секунды, а когда он закончится, выполняем первую функцию из таблицы (подготовка к полету)
Timer.callLater(2, function () action[curr_state]() end)
```

Попробуйте написать код для большего числа точек, чтобы получить более плавный полет.





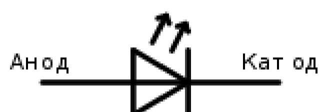
ФОНД СОДЕЙСТВИЯ РАЗВИТИЮ ВОЕННОГО ОБРАЗОВАНИЯ

Занятие 13. Программирование на языке Lua – Управление светодиодной панелью

Теория:

Светодиод – полупроводниковый прибор, который излучает практически монохромный свет в прямом направлении (монохромный – очень узкий диапазон света, т.е. только волны определенного цвета).

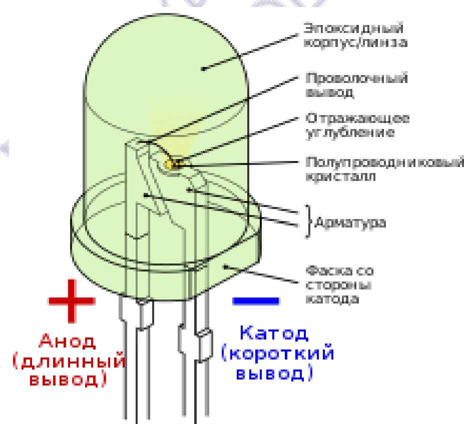
В электрических схемах светодиод обозначается следующим образом



Как вы видите, у светодиода есть полярность, это все из-за его полупроводниковой структуры.

Полупроводник – это общее название группы веществ, которые имеют разную электрическую проводимость в зависимости от температуры (очень холодно – почти не проводят ток, тепло – хорошо проводят). Нахождение полупроводников значительно повлияло на развитие электроники, так как был изобретен полупроводниковый диод – прибор, пропускающий ток только в одну сторону (например, преобразует переменный ток в постоянный).

У стандартного светодиода две ноги – плюс (анод) и минус (катод).



Похожая картина у RGB светодиода

ФОНД СОДЕЙСТВИЯ РАЗВИТИЮ ВОЕННОГО ОБРАЗОВАНИЯ

У него 4 ноги, 1 катод и 3 анода, для каждого из цветов. Соответственно, подавая разный ток на каждую из ног светодиода можно получить любой цвет видимого спектра.

На предыдущих занятиях мы уже работали со светодиодами, но на этом будем использовать дополнительный LED модуль.

Он представляет из себя 25 светодиодов, пронумерованные с верхнего левого угла платы.

Для инициализации светодиодов используется класс Ledbar, который имеет 2 основные функции – инициализацию (Ledbar:new(<Count>), где Count – число светодиодов) и (Ledbar:set(x, r, g, b), где x – номер светодиода)

Практика:

1. Вывод чисел от 1 до 10

В начале работы инициализируем светодиоды и пропишем функцию callback, которую пока что оставим пустой.

```
-- Указываем количество светодиодов
local ledbar = Ledbar.new(29)

-- переменная текущего состояния
local curr_state = "START"

-- 4 светодиода на базовой плате
local matrixLeds = 4

-- 25 светодиодов на доп. модуле
local modulLeds = 25

-- Используется для распаковки массива
local unpack = table.unpack
```

```
-- функция обработки собы
function callback(event)
end
```

После этого необходимо написать список, в котором будут все используемые числа по отдельности.

```
-- Массив для вывода чисел от 1 до 10
local arrayOfNumbers = {
    {12,8,4,9,14,19,24},
    {6,2,3,4,10,14,13,17,21,22,23,24,25},
    {1,2,3,4,5,9,13,19,25,24,23,22,21},
    {2,4,7,9,12,13,14,19,24},
    {5,4,3,2,1,6,11,12,13,14,15,20,25,24,23,22,21},
    {4,8,12,13,14,17,19,22,23,24},
    {2,3,4,5,10,14,18,23},
    {2,3,4,7,9,13,17,19,22,23,24},
    {2,3,4,7,9,12,13,14,19,24,23,22},
    {1,6,11,16,21,3,4,5,8,10,13,15,18,20,23,24,25},
}
```

В этом списке на позиции один светодиоды, которые будут загораться при цифре 1, на второй – 2 и т.д.

Также немного модернизируем таблицу цветов, чтобы вместо colors[<номер цвета>] нужно было вводить colors.<цвет> (например colors.green).

ФОНД СОДЕЙСТВИЯ РАЗВИТИЮ ИНЖЕНЕРНОГО ОБРАЗОВАНИЯ

```lua
-- RGB коды основных цветов
local colors = {
    red = {1,0,0},
    green = {0,1,0},
    blue = {0,0,1},
    purple = {1, 0, 1},
    cyan = {0, 1, 1},
    yellow = {1, 1, 0},
    white = {1, 1, 1},
    black = {0, 0, 0}
}
```

Теперь напишем функцию для вывода любой цифры на дополнительный модуль.

Создадим функцию (local function()), в которой будет цикл от 1 до длины вложенного списка с номерами светодиодов. Далее с помощью ledbar:set() включаем нужные светодиоды.

```lua
-- Функция для вывода числа от 0 до 10 на доп. модуль
function numberLed(number, color)
    for i = 1, #arrayOfNumbers[number] do
        ledbar:set(arrayOfNumbers[number][i]+matrixLeds-1, unpack(color))
    end
end
```

Нумерация светодиодов на доп. модуле идет после основной платы, что и пропишем внутри set() -> arrayOfNumbers[number][i] – номер каждого светодиода из списка, к которому добавляем число светодиодов на основной плате (matrixLeds = 4) и вычитаем 1, так как нумерация на основной плате идет не с 1, а с 0 (суммарная разница между нумерациями получается +3).

Чтобы запустить функцию, в конце программы нужно прописать

```lua
numberLed(1, colors.green)
```

где первый параметр это число, которое выведем, а второй – цвет.

Чтобы выводить несколько цифр подряд необходимо выключать все светодиоды между ними, в ней будет цикл, который проходит по всем светодиодом и меняет их цвет на черный.

```
-- Функция выключения всех светодиодов
function allLedsOff()
    for i = 0, matrixLeds+modulLeds-1 do
        ledbar:set(i, unpack(colors.black))
    end
end
```

Вот пример от 1 до 3.

```
numberLed(1, colors.green)
sleep(1)
allLedsOff()
sleep(1)
numberLed(2, colors.green)
sleep(1)
allLedsOff()
sleep(1)
numberLed(3, colors.green)
sleep(1)
allLedsOff()
sleep(1)
```

Либо с помощью цикла, что будет правильнее

```
for k=1, 10 do
    numberLed(k, colors.green)
    sleep(1)
    allLedsOff()
    sleep(1)
end
```

Попробуйте самостоятельно, с помощью цикла внутри цикла реализовать смену цветов.

Занятие 14. Программирование на языке Lua – Таймеры и асинхронность

Теория:

Во многих языках программирования большинство функций выполняются друг за другом. Это называется синхронным выполнением. Т.е. каждое действие начинается, только когда завершится предыдущее.

Синхронное выполнение имеет ряд плюсов, когда все действия следуют друг за другом никогда не произойдет ошибки выполнения (допустим, один и тот же светодиод сначала загорается красным, а затем синим, а не сразу фиолетовый). Но следует понимать, что если хоть одна функция не выполнится до конца, то полностью встанет работа всей оставшейся программы.

Асинхронное выполнение запускает сразу несколько процессов параллельно друг другу. Оно так же имеет ряд своих плюсов и минусов. О них вы можете узнать, более углубленно погрузившись в работу ядер компьютера.

Для нас же важно, что с помощью асинхронных функций Пионер может выполнять несколько действий одновременно.

Timer – объект, который позволяет с неким периодом запускать различные функции. Он имеет несколько методов: Timer.new(sec, func), где sec – период запуска функции, а func – фукнция, которая будет запускаться.

Две функции необходимы для работы таймера – start и stop (<имя таймера>:start() и <имя таймера>:stop() соответственно).

Помимо этого, есть пара функций, callLater(delay, func), которая запускает функцию через какое-либо время. Остаются еще 2 функции, которые используются редко, их вы можете найти в официальном API.

Практика:

1. Бесконечный полет по кругу с одновременным поворотом по ходу движения.

Как вы заметили из предыдущих занятий, когда мы ставим поворот движение к точке и рыскание, обе эти команды выполняются синхронно, т.е. одна строго после второй. В случае полёта по квадрату это наоборот удобно, так как перед каждой стороной квадрата дрон поворачивается к ней «лицом».

Однако при движении по окружности это будет не столь красиво, так что необходимо написать два таймера, для движения и поворота по окружности.

Для начала инициализируем светодиоды, пропишем callback(events) и пропишем начальный state.

```
-- количество светодиодов на основной плате пионера
local ledNumber = 4
-- создание порта управления светодиодами
local leds = Ledbar.new(ledNumber)
-- ассоциируем функцию распаковки таблиц из модуля table для упрощения
local unpack = table.unpack

-- переменная текущего состояния
local curr_state = "PREPARE_FLIGHT"

-- функция, изменяющая цвет 4-х RGB светодиодов на основной плате пионера
local function changeColor(color)
    -- проходим в цикле по всем светодиодам с 0 по 3
    for i=0, ledNumber - 1 do
        leds:set(i, unpack(color))
    end
end

-- таблица цветов в формате RGB для передачи в функцию changeColor
local colors = {
        {1, 0, 0}, -- красный
        {1, 1, 1}, -- белый
        {0, 1, 0}, -- зеленый
        {1, 1, 0}, -- желтый
        {1, 0, 1}, -- фиолетовый
        {0, 0, 1}, -- синий
        {0, 0, 0}  -- черный/отключение светодиодов
}
```


ФОНД СОДЕЙСТВИЯ РАЗВИТИЮ ВОЕННОГО ОБРАЗОВАНИЯ

```
-- функция обработки событий, автоматически вызывается автопилотом
function callback(event)
    -- если достигнута необходимая высота, то выполняем функцию из таблицы, соответствующую текущему состоянию
    if (event == Ev.TAKEOFF_COMPLETE) then
        action[curr_state]()
    end

    -- если коптер с чем-то столкнулся, то зажигаем светодиоды красным
    if (event == Ev.SHOCK) then
        changeColor(colors[1])
    end

    -- если пионер достигнул точки, то выполняем функцию из таблицы, соответствующую текущему состоянию
    if (event == Ev.POINT_REACHED) then
        action[curr_state]()
    end

    -- если пионер приземлился, то выключаем светодиоды
    if (event == Ev.COPTER_LANDED) then
        changeColor(colors[7])
    end

end
```

Создадим таймер для полета по окружности. Он во многом будет повторять код из позапрошлого занятия.

Назовем таймер pointT, большая T будет показывать, что это объект таймера.

pointT = Timer.new(0.1, function()
<тело таймера>
end)

В теле нам необходимы функции:

1. Увеличения угла
2. Расчет координат полета
3. Собственно полет

angle = angle + value это функция 1, которая увеличит угол на значение

Расчет координат проводим так же, как и на предыдущих занятиях.

Функция полета – *ap.goToLocalPoint(x,y,z)*

Также поставим обнуление угла при >360, чтобы получить бесконечный полет.

ФОНД СОДЕЙСТВИЯ РАЗВИТИЮ ВОЕННОГО ОБРАЗОВАНИЯ

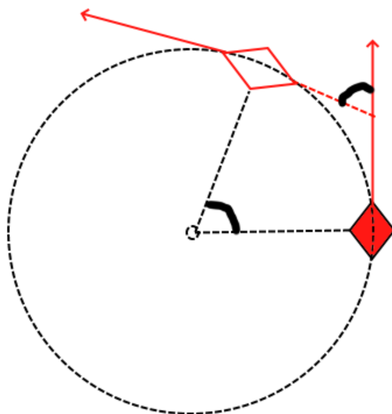
```
local r = 0.3
local angle = 0
local xCord = 0
local yCord = 0
local height = 0.7
local value = 3

pointT = Timer.new(0.1, function()
    angle = angle + value
    if angle > 360 then
        angle = 0
    end
    yCord = r*math.sin(angle * math.pi / 180)
    xCord = r*math.cos(angle * math.pi / 180)
    ap.goToLocalPoint(xCord, yCord, height)
end)
```

Переменные – r – радиус окружности, angle – начальный угол, value – значение угла, с которым будет проходить шаг по окружности.

Аналогичным образом напишем таймер для поворота Пионера по ходу движения.

```
local i = 0

angleT = Timer.new(0.1, function()
    ap.updateYaw(-i/180*math.pi)
i = i+value
end)
```

Угол поворота будет соответствовать value из первого таймера, это можно понять из следующей схемы.



Теперь пропишем действия.

ФОНД СОДЕЙСТВИЯ РАЗВИТИЮ ИНЖЕНЕРНОГО ОБРАЗОВАНИЯ

Первое (взлет) возьмем из предыдущих занятий.

```
["PREPARE_FLIGHT"] = function(x)
    changeColor(colors[2]) -- смена цвета свет
    Timer.callLater(2, function () ap.push(Ev.
    Timer.callLater(4, function () changeColor
    Timer.callLater(6, function ()
        ap.push(Ev.MCE_TAKEOFF) -- еще через 2
        ap.goToLocalPoint(0,0, height)
        curr_state = "FLIGHT_TO_FIRST_POINT" -
    end)
end,
```

В следующем действии нужно запустить оба таймера и прописать команду sleep, которая будет останавливать действие скрипта на нужное нам время.

```
["FLIGHT_TO_FIRST_POINT"] = function (x)
    changeColor(colors[4]) -- смена цвета светодиодов на желтый
    Timer.callLater(1, function ()
        angleT:start()
        pointT:start()
        sleep(6) -- секунд выполнения цикла
        curr_state = "PIONEER_LANDING" -- переход в следующее состояние
    end)
end,
```

Последним действием пропишем остановку таймеров, возврат в начальную точку (если необходимо) и посадку.

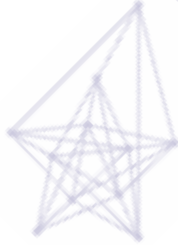
```
["PIONEER_LANDING"] = function (x)
    changeColor(colors[2]) -- смена цвета светодиодов на белый
    Timer.callLater(2, function ()
        pointT:stop()
        angleT:stop()
        ap.goToLocalPoint(0, 0, height)
        ap.push(Ev.MCE_LANDING) -- отправка команды автопилоту на посадку
    end)
end
```

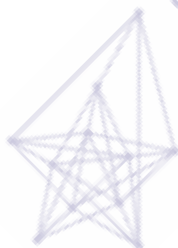
Ради безопасности пропишем также остановку таймеров при столкновении дрона с объектом, это делается в функции callback.

```
-- если коптер с чем-то столкнулся, то зажигаем светодиоды красным
if (event == Ev.SHOCK) then
    changeColor(colors[1])
    angleT:stop()
    pointT:stop()
end
```

И в конце программы напишем код, запускающий выполнение.

```
-- включаем светодиод (красный цвет)
changeColor(colors[1])
-- запускаем одноразовый таймер на 2 секунды, а когда он закончится, выполняем первую функцию из таблицы (подготовка к полету)
Timer.callLater(2, function () action[curr_state]() end)
```





Занятие 15. Класс Sensors. Получение данных от автопилота.

Теория:

Квадрокоптер имеет множество систем, помогающих в полёте. С некоторыми, такими как данные с пульта управления, мы уже работали в уроках, связанных с модулем захвата груза.

Для всех этих данных используется отдельный класс – Sensors.

Получение данных из этого класса имеет общий паттерн – объявить переменную и с помощью таймера обновлять в ней значения.

Список всех параметров, которые можно получать, есть в разделе документации на сайте.

В рамках программы по программированию наиболее частым будет использование переменной *Sensors.rc*, которая позволяет получать данные от пульта (rc – radio channel). Эта переменная представляет собой массив из 8 значений, каждое из которых соответствует своему каналу.

Практика:

1. Одновременный запуск программы на нескольких коптерах.

Одним из преимуществ работы с тумблерами является возможность синхронного запуска любой программы автономного полёта с помощью одного пульта.

Для этого необходимо сначала проверить, настроен ли пульт, а именно 8 канал, а также забиндить несколько коптеров на один пульт.

Далее, после стандартной полетной программы следует добавить следующий код:

ФОНД СОДЕЙСТВИЯ РАЗВИТИЮ ВОЕННОГО ОБРАЗОВАНИЯ

```lua
local rc = Sensors.rc
timer = Timer.new(0.1, function()
    _,_,_,_,_,_,_,ch8 = rc()
    if ch8>0 then
        leds:set(0,1,0,0)
    elseif ch8<0 then
        timer:stop()
        -- <начало полетной программы>
    end
end)

timer:start()
```

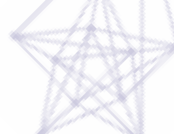
Данный код очень похож на код запуска магнита, мы также сначала объявляем переменную *rc*, затем создаем таймер, который считывает данную переменную, и в зависимости от положения тумблера выполняет разные части кода. Ch8 > 0 – тумблер SwA поднят вверх, Ch8 < 0 – тумблер опущен.

Вместо <начало программы> необходимо вставить код, который запускает Пионер, допустим, если программа создана через TRIK – это будет *action[curr_state]()*, если код написан сразу в Pioneer Station, это строки предполетной подготовки двигателей, взлет и тд.

Помимо этого, в автономном полёте не задействован тумблер SwD, который настроен на 6 канале. Единственное отличие кода будет в том, что тумблер SwD имеет значения 1 (вверх) и 0 (вниз), а не 1 (вверх) и -1 (вниз), как у тумблера SwA.

Далее предлагается самостоятельно написать программу поискового фонаря.

С помощью двух тумблеров запрограммировать квадрокоптер так, чтобы один тумблер отвечал за моргание светодиодами с какой-либо частотой, а второй тумблер включал всю панель светодиодов.



ФОНД СОДЕЙСТВИЯ РАЗВИТИЮ ВОЕННОГО ОБРАЗОВАНИЯ

Занятие 16. Программирование на Lua – «Формула 1»

Теория:

На этом занятии мы усовершенствуем работу с полетом по кругу, добавив счётчик кругов.

Практика:

1. Бесконечный полет по кругу с использованием тумблера.

Для начала необходимо написать код, позволяющий прервать полет по кругу с предыдущего занятия.

Для этого необходимо инициализировать переменную rc, с помощью команды

```
local rc = Sensors.rc
```

Далее создадим таймер, который будет проверять состояние тумблера SwA, и, если он в вертикальном положении, продолжать полет по кругу, а если опустить, посадит Пионера.

```
checkT = Timer.new(0.1, function () --
    _, _, _, _, _, _, _, ch8 = rc() --
    if(ch8 < 0) then  -- если сигнал с
        changeColor(colors[5])
        pointT:start()
        angleT:start()
    else -- если сигнал с пульта 1 (SWA
        changeColor(colors[6])
        pointT:stop()
        angleT:stop()
        curr_state = "PIONEER_LANDING"
    end
end)
```

Данный код аналогичен коду из предыдущего занятия.

Далее, в основном стейте прописываем старт таймера, а в стейте Pioneer_Landing отключаем его.

```lua
["PREPARE_FLIGHT"] = function(x)
    changeColor(colors[2]) -- смена цвета светодиодов на белый
    Timer.callLater(2, function () ap.push(Ev.MCE_PREFLIGHT) end)
    Timer.callLater(4, function () changeColor(colors[3]) end)-- €
    Timer.callLater(6, function ()
        ap.push(Ev.MCE_TAKEOFF) -- еще через 2 секунды (суммарно ч
        ap.goToLocalPoint(0,0, height)
        curr_state = "FLIGHT_TO_FIRST_POINT" -- переход в следующе
    end)
end,
["FLIGHT_TO_FIRST_POINT"] = function (x)
    changeColor(colors[4]) -- смена цвета светодиодов на желтый
    Timer.callLater(1, function ()
        checkT:start()
         -- переход в следующее состояние
    end)
end,
["PIONEER_LANDING"] = function (x)
    changeColor(colors[2]) -- смена цвета светодиодов на белый
    Timer.callLater(2, function ()
        checkT:stop()
        ap.goToLocalPoint(0, 0, height)
        ap.push(Ev.MCE_LANDING) -- отправка команды автопилоту на
    end)
end
```

2. Счетчик пройденных кругов.

За основу возьмем код, использованный в одном из предыдущих занятий.

Для начала изменим количество светодиодов в начале программы (вместе с LED модулем их 29).

Далее необходимо создать список, в котором будут числа от 1 до 10.

```lua
local arrayOfNumbers = { -- числа от 1 до 10
    {12,8,4,9,14,19,24},
    {6,2,3,4,10,14,13,17,21,22,23,24,25},
    {1,2,3,4,5,9,13,19,25,24,23,22,21},
    {2,4,7,9,12,13,14,19,24},
    {5,4,3,2,1,6,11,12,13,14,15,20,25,24,23,22,21},
    {4,8,12,13,14,17,19,22,23,24},
    {2,3,4,5,10,14,18,23},
    {2,3,4,7,9,13,17,19,22,23,24},
    {2,3,4,7,9,12,13,14,19,24,23,22},
    {1,6,11,16,21,3,4,5,8,10,13,15,18,20,23,24,25}
}
```

Далее создадим переменную, считающую количество кругов.

```
local laps = 1
```

Эту переменную будем увеличивать в таймере point, внутри if angle>360, это как раз тот момент, когда коптер пролетел полный круг.

```
angle = angle + value
if angle >= 360 then
    angle = angle - 360
    laps = laps + 1
end
```

Далее пропишем отдельный таймер для вывода числа кругов.

```
local laps = 1
lapT = Timer.new(1, function()
    changeColor(colors.black)
    sleep(0.1)
    numberLed(laps, colors.green)
end)
```

Каждую секунду будем обновлять вывод числа кругов.

И, в основной функции также запустим этот таймер

```
["FLIGHT_TO_FIRST_POINT"] = function (x)
    changeColor(colors.yellow) -- смена цве
    Timer.callLater(1, function ()
        checkT:start()
        lapT:start()
         -- переход в следующее состояние
    end)
end,
["PIONEER_LANDING"] = function (x)
    changeColor(colors.white) -- смена цвет
    Timer.callLater(2, function ()
        checkT:stop()
        ap.goToLocalPoint(0, 0, height)
        lapT:stop()
        ap.push(Ev.MCE_LANDING) -- отправка
    end)
end
```


ФОНД СОДЕЙСТВИЯ РАЗВИТИЮ ИНЖЕНЕРНОГО ОБРАЗОВАНИЯ

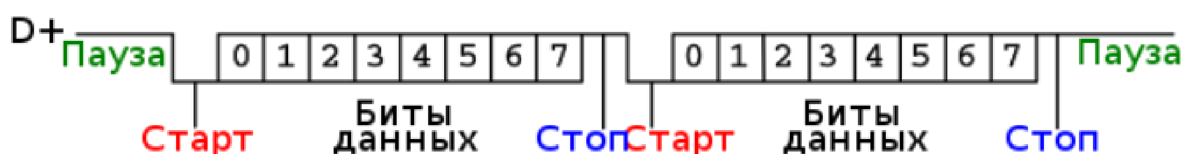
Занятие 17. Программирование на языках Lua и Python – Программируемая камера OpenMV.

Теория:

Python – популярный язык программирования, который используется во многих сферах жизни (от написания сайтов и игр до приложений по 3D-моделированию). Он достаточно прост и хорошо документирован, что позволяет быстро освоить его основные принципы.

Python хорошо подходит для потоковой обработки видео-сигнала.

Многие модули (модуль захвата груза, камера OpenMV) общаются с платой Пионера посредством UART канала связи. Это основной канал связи, используемый в небольших платах, так как он занимает только один цифровой канал. Вся информация кодируется в по стандартизированным схемам в двоичный код, и затем передается по каналу с определенной частотой, называемой «битрейт» (число бит в секунду) или «бодрейт (baudrate). На Пионере частота составляет 9600, т.е. 9600 байт в секунду.



Напомню, что 1 байт = 8 бит, 1 килобайт (Кб) = 1024 байт, 1 мегабайт (Мб) = 1024 Кб, и т.д.

В начале и в конце каждой посылки (на примере указаны 8-битовые посылки – самый частый вариант) указывается специальный бит начала полезных данных и завершения. Таким образом, полезная частота передачи данных составляет 80% от бодрейта, соответственно для Пионера она будет равна 7680 бит в секунду, или порядка 0,94 Мб в секунду.

Этого хватает, чтобы в режиме реального времени передавать, а значит и обрабатывать видео в 1080p (Full HD) 30fps.

Также UART часто имеет бит четности (parity bit). Он необходим, как проверка того, что сигнал идет без прерываний и ошибок. Если битрейт UART'а выставлен неправильный или по каким-либо причинам бит начала или конца передачи не передался, весь следующий поток данных может сдвинуться на бит, что приведет к

полной потере данных (как пример – 0010010 – это число 10 в двоичной системе, при потере любого нуля может произойти сдвиг, допустим 001010 – это число 8). Поэтому существует возможность занять еще один бит в конце каждого блока данных, который будет равен 1, если число единиц в блоке нечетное, и 0, если четное (бит четности). Это будет сигналом программе, что поток данных непрерывен.

UART – двусторонний канал связи, плата помимо приема данных с камеры может и передавать сигнал (как с модулем захваты груза, где плата включала и выключала электромагнит). Единственное, канал может либо передавать данные, либо принимать, так как UART занимает один цифровой канал.

Практика:

1. Яркость предмета.

Для работы с камерой OpenMV необходимо скачать OpenMV IDE.

OpenMV это портированная библиотека OpenCV, предназначенная для небольших устройств, что идеально подходит для Пионера.

ФОНД СОДЕЙСТВИЯ РАЗВИТИЮ ВОЕННОГО ОБРАЗОВАНИЯ

Так как в рамках занятий Python не изучался, ниже прилагаю сразу весь код на OpenMV, которым будем пользоваться.

```
from pyb import UART, LED

import sensor, lcd, image, time, utime

ledBlue = LED(2)
ledGreen = LED(3)

ledBlue.on()
sensor.reset()                           # Перезапустить камеру
sensor.set_pixformat(sensor.RGB565) # Ставим формат камеры RGB565
sensor.set_framesize(sensor.LCD)    # Выставляем размеры видео 320x240
sensor.skip_frames(100)      # Пауза необходима для выполнения настроек
clock = time.clock()                     # Создаем объект для отслеживания FPS
lcd.init()
#lcd.set_backlight(True)
ledBlue.off()

#Init uart

uart = UART(3)
uart.init(9600, bits=8, parity=None, stop=1, timeout_char=1000) # Инициализируем UART

M_LED_COUNT = 10
led_counter = M_LED_COUNT
led_mode = 0
while(True):
   clock.tick()                        # Update the FPS clock.
   clk = utime.ticks_ms()
   img = sensor.snapshot()             # Получить картинку и записать ее в переменную
   #print(clock.fps())

   for r in img.find_rects(threshold = 40000):
       img.draw_rectangle(r.x(), r.y(), r.w(), r.h(), (255, 0, 0))
       for p in r.corners():
           img.draw_circle(p[0], p[1], 5, color = (0, 255, 0))
       print(r)

   lcd.display(img)

   print(img.get_histogram().get_statistics().l_mean())
   uart.writechar(img.get_histogram().get_statistics().l_mean())
   led_counter = led_counter - 1
   if (led_counter == 0):
       if (led_mode == 0):
           ledGreen.on()
       else:
           ledGreen.off()
       led_mode = 1 - led_mode
       led_counter = M_LED_COUNT
   while (clk + 100 > utime.ticks_ms()):
       pass
```

В начале программы импортируем все необходимые библиотеки, далее инициализируем светодиоды платы. RGB565 означает, что красный и синий цвет кодируются 5 битами, а зеленый 6 битами.

В инициализации UART указываем все параметры, 9600 – baudrate, parity – бит четности.

Далее идет код, получающий картинку с камеры и обрабатывающий ее.

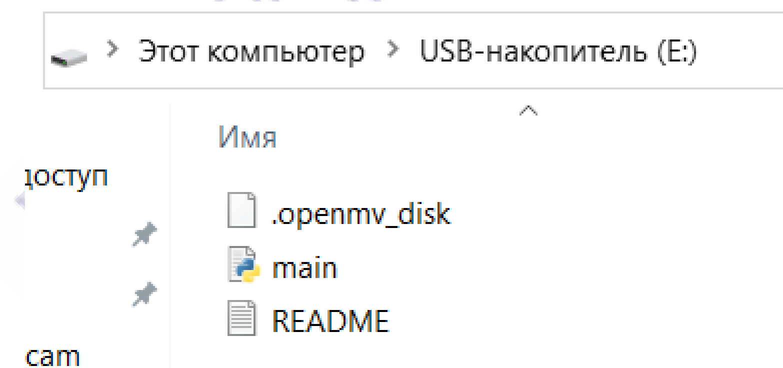
Камеру необходимо подключить отдельно с помощью собственного micro USB, подключить камеру и загрузить скрипт

```
    clk = utime.ticks_ms()
    img = sensor.snapshot()         # Получить
    #print(clock.fps())

    for r in img.find_rects(threshold = 40000):
        img.draw_rectangle(r.x(), r.y(), r.w(),
        for p in r.corners():
            img.draw_circle(p[0], p[1], 5, colo
        print(r)

    lcd.display(img)

    print(img.get_histogram().get_statistics().
    uart.writechar(img.get_histogram().get_stat
    led_counter = led_counter - 1
    if (led_counter == 0):
        if (led_mode == 0):
            ledGreen.on()
        else:
            ledGreen.off()
        led_mode = 1 - led_mode
        led_counter = M_LED_COUNT
    while (clk + 100 > utime.ticks_ms()):
        pass
```

Соединить Ctrl+E

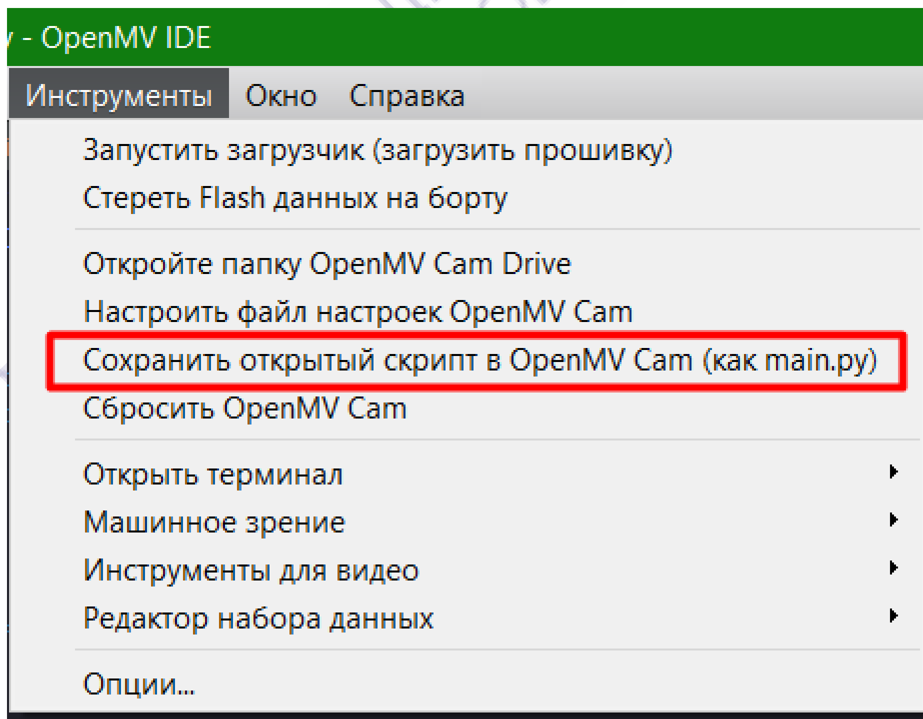
Результаты поиска   Последовательный терминал

Обратите внимание, если вы откроете файлы камеры, когда она подключена («Мой компьютер» -> «USB-накопитель»),



то там будет один python файл (main.py). Именно в этот main.py файл необходимо загрузить код. Если вы будете создавать новый файл в OpenMV IDE, то необходимо загрузить этот файл в камеру, иначе он будет работать только пока камера подключена к компьютеру.

Это можно сделать так, вкладка «Инструменты», «Сохранить открытый скрипт»



Теперь пропишем код для Пионера.

В Pioneer Station инициализируем светодиоды

```
-- количество светодиодов на основной плате пионера
local ledNumber = 4
-- создание порта управления светодиодами
local leds = Ledbar.new(ledNumber)

-- функция, изменяющая цвет 4-х RGB светодиодов на основной плате пионера
local function changeColor(red, green, blue)
    for i=0, ledNumber - 1, 1 do
        leds:set(i, red, green, blue)
    end
end
```

В функции callback напишем только условие посадки при низком заряде аккумулятора.

```
-- определяем функцию анализа возникающих событий в системе
function callback(event)
    -- проверка, низкое ли напряжение на аккумуляторе
    if (event == Ev.LOW_VOLTAGE2) then
        emergency()
    end
end
```

Перед callback пропишем саму функцию emergency.

В ней будем останавливать действие таймера, который пропишем чуть позднее.

```
-- функция, которая меняет цвет светодиодов на красный и выключает таймер
local function emergency()
   takePhotoTimer:stop()
   -- так как после остановки таймера его функция выполнится еще раз, то меняем цвета светодиодов на красный через секунду
   Timer.callLater(1, function () changeColor(1, 0, 0) end)
end
```

Теперь необходимо прописать все переменные для инициализации UART.

```
-- инициализируем Uart интерфейс
local uartNum = 4 -- номер Uart интерфейса (USART4)
local baudRate = 9600 -- скорость передачи данных
local dataBits = 8
local stopBits = 1
local parity = Uart.PARITY_NONE
local uart = Uart.new(uartNum, baudRate, parity, stopBits) -- создание протокола обмена
```

Далее прописываем функцию приема данных

```
local N = 10
local i = 7
local strUnpack = string.unpack
function getData() -- функция приёма байта данных
   i = i + 1
   if (i == N + 1) then i = 0 end
   buf = uart:read(uart:bytesToRead()) or '0'
   if (#buf == 0) then buf = '\0' end
   leds:set(1, 0, i/N, 0.5 - 0.5*i/10)
   if (strUnpack ~= nil) then
      local b = strUnpack("B", buf)
      return b -- примерно должно так работать
   else
      return 20
   end
end
```

Она читает пакет данных из камеры и выставляет значение яркости на светодиоде.

И в конце прописываем таймер и запускаем его

```
local takerFunction = function () -- функция для периодического чтения данных из UART
 local intensity = getData() / 100.0
 changeColor(intensity, intensity, intensity)
end
local interval = 0.1
getMeasureTimer = Timer.new(interval, takerFunction) -- таймер для создания фото
getMeasureTimer:start()
```

GEOSCAN

Методическое пособие является интеллектуальной собственностью ООО «Геоскан».
2021