

UAVPROF DRONE SIMULATOR

Autonomous

Инструкция по эксплуатации

Настоящая документация может быть использована только для поддержки работоспособности продуктов, установленных на основании договора с ООО «СТРАТУС». Документация может быть передана на основании договора, по которому производится (производилась или будет производиться) установка продуктов, или явно выраженного согласия ООО «СТРАТУС» на использование данной документации. Если данный экземпляр документации попал к вам каким-либо иным образом, пожалуйста, сообщите об этом в ООО «СТРАТУС» по адресу, приведенному ниже.

Все примеры, приведенные в документации (в том числе примеры отчетов и экранных форм), составлены на основании тестовой базы ООО «СТРАТУС». Любое совпадение имен, фамилий, названий компаний, банковских реквизитов и другой информации с реальными данными является случайным.

Все встречающиеся в тексте торговые знаки и зарегистрированные торговые знаки являются собственностью их владельцев и использованы исключительно для идентификации программного обеспечения или компаний.

Данная документация может не отражать некоторых модификаций программного обеспечения. Если вы заметили в документации ошибки или опечатки или предполагаете их наличие, пожалуйста, сообщите об этом в ООО «СТРАТУС».

Все имущественные авторские права сохраняются за ООО «СТРАТУС» в соответствии с действующим законодательством.

© ООО «СТРАТУС», 2024

ООО «СТРАТУС»

119607, г. Москва, вн.тер.г. Муниципальный округ Раменки, б-р Раменский, д. 1

Тел.: +7 (931) 604-34-33

Электронная почта: sim@uavprof.com

Содержание

1. Общие сведения	3
1.1. Модели аппаратов	3
1.1.1. Общая информация для всех аппаратов	3
1.1.2. Камера	3
1.2. Системы координат	4
1.3. Алгоритм управления	4
1.3.1. Управляющие воздействия	4
1.3.2. Изменение коэффициентов ПИД-регуляторов автопилота	5
1.4. Формации	5
1.5. Использование камеры аппарата	5
1.6. Горячие клавиши переключения видов	6
1.7. Переключение на Vulkan	6
2. Начало работы	7
3. Дисциплины	8
3.1. Синхронный полёт	8
3.1.1. Гоночная трасса	8
3.1.2. Формации	9
3.1.3. Задание	10
3.1.4. Запуск	11
3.1.5. Критерии оценивания	11
3.2. Командная гонка	11
3.2.1. Гоночная трасса	11
3.2.2. Аппарат	14
3.2.3. Задание	14
3.2.4. Запуск	14
3.2.5. Критерии оценивания	14
3.3. Уход от столкновения	14
3.3.1. Гоночная трасса	15
3.3.2. Задание	16
3.3.3. Запуск	17
3.3.4. Критерии оценивания	17
4. Работа со светодиодной панелью	19
4.1. API модуля	19
4.2. Разбор примера	19
4.3. Запуск	20
4.4. Дополнительные примеры	22
4.4.1. Управление одним светодиодом	22
4.4.2. Цвета	23
4.4.3. Цифры	23
4.4.4. Смайлики	24
История изменений	26

1. Общие сведения

UAVPROF Drone Simulator: Autonomous — это обучающий продукт (далее — Симулятор) для разработки алгоритмов управления дронами.

Продукт позволяет использовать его для двух уровней сложности — студенческого и профессионального.

Для запуска продукта используется командная строка, а ПО пользователя подключается к автопилоту по сети через MAVLink по UDP и к Симулятору через ROS 2.

1.1. Модели аппаратов

В зависимости от дисциплины и уровня сложности, используется одна или группа из следующих моделей БПЛА мультироторного типа:

- FPV F5D Auton;
- FPV F5D Auton GPS (доступна GPS);
- FPV F5D Auton LPS (доступна LPS — система локального позиционирования).

Наборы параметров автопилотов указаны в файлах, расположенных в директории params соответствующей версии Симулятора (<https://gitflic.ru/project/uavprofsim/parma-bas/file?file=docs%2Fparams>):

- fpv_f5d_auton.params — для FPV F5D Auton;
- fpv_f5d_auton_gps.params — для FPV F5D Auton GPS;
- fpv_f5d_auton_lps.params — для FPV F5D Auton LPS.

Описание параметров автопилота см. https://docs.px4.io/v1.13/en/advanced_config/parameter_reference.html.

1.1.1. Общая информация для всех аппаратов

- Масса: 0.696 кг;
- размер: 0.263 x 0.316 x 0.089 м;
- доступна [камера](#);
- PX4 1.13.2.

Табл. 1. Координаты пропеллеров относительно центра масс

№	x, м	y, м	z, м
1	0.068	-0.094	0.010
2	-0.068	0.082	0.010
3	0.068	0.094	0.010
4	-0.068	-0.082	0.010

1.1.2. Камера

На всех моделях аппаратов доступна камера со следующими характеристиками:

- цветное изображение (RGB);
- вертикальный угол обзора (VFOV) — 86.8 градусов;

- разрешение 1280x720 пикселей;
- максимальное количество кадров в секунду (FPS) – 60;
- изображения передаются через топик ROS 2;
- координаты относительно центра масс: (0.107 м, 0.000 м, 0.000 м);
- угол наклона (тангаж): 20 градусов.

1.2. Системы координат

Используются следующие системы координат:

- локальная система координат автопилота;
- локальная система координат Симулятора;
- глобальная система координат ГНСС.

Базисы, соответствующие осям X, Y, Z локальных систем координат:

- ENU (Восток-Север-Вверх) для локальной системы координат Симулятора, сообщений ROS/MAVROS;
- NED (Север-Восток-Вниз) для автопилота PX4 и сообщений MAVLink.

Точке отсчета (0, 0, 0) локальной системы координат Симулятора соответствует точка (Широта, Долгота, Высота) глобальной системы координат, задаваемая в JSON-файле параметров (свойства `latitude`, `longitude`, `altitude` объекта `/session/world`). Все координаты объектов в текущем документе заданы в базисе ENU (Восток-Север-Вверх) локальной системы координат Симулятора.

1.3. Алгоритм управления

Управление каждым аппаратом в группе происходит с помощью алгоритма, разрабатываемого пользователем.

Управляющие воздействия на аппараты выдаются в одном из следующих вариантов:

- по точкам в локальной системе координат;
- по точкам в глобальной системе координат;
- по скоростям в локальной и глобальной системах координат;
- по ускорениям;
- по тяге и ориентации/угловым скоростям аппарата.

Алгоритмы должны быть или уникальными или иметь принципиальные отличия от общедоступных.

1.3.1. Управляющие воздействия

Программное управление автопилотом извне выполняется с помощью трех MAVLink-сообщений:

- [SET_POSITION_TARGET_LOCAL_NED](#);
- [SET_POSITION_TARGET_GLOBAL_INT](#);
- [SET_ATTITUDE_TARGET](#).

Сообщения ROS/MAVROS, соответствующие MAVLink-сообщениям:

- [PositionTarget](#);
- [GlobalPositionTarget](#);
- [AttitudeTarget](#).

Код обработки на стороне автопилота (для справки и уточнения деталей реализации):

- [SET_POSITION_TARGET_LOCAL_NED](#);
- [SET_POSITION_TARGET_GLOBAL_INT](#);
- [SET_ATTITUDE_TARGET](#).

Первое сообщение позволяет управлять в локальной системе координат по точкам, скоростям, ускорениям, рысканию и скорости рыскания.

Второе — в глобальной системе координат по точкам (широта, долгота, высота), скоростям, ускорениям, рысканию и скорости рыскания.

Третье — по положению в пространстве (кватерниону), скоростям крена, тангажа и рыскания, нормированному значению тяги.

В каждом сообщении возможны разные комбинации управления, которые задаются специальным полем-маской. Не все комбинации могут приниматься автопилотом, для уточнения необходимо обращаться к исходным кодам (см. выше). Примеры реализации управления по точкам и скоростям в локальной системе координат представлены в файлах:

- `ros1-group` — пример управления группой аппаратов через ROS;
- `ros2` — пример для студенческой гонки;
- `ros2-group` — пример управления группой аппаратов через ROS 2.

1.3.2. Изменение коэффициентов ПИД-регуляторов автопилота

Значения коэффициентов ПИД-регуляторов автопилота выведены в соответствующие параметры (описание параметров см. в разделе [Модели аппаратов](#)). Описание ПИД-регуляторов и рекомендации по их изменению представлены в Multicopter PID Tuning Guide [Basic](#) и [Advanced](#).

1.4. Формации

Формации представляют собой совокупность точек пространства, в каждой из которых должен находиться один из аппаратов (любой аппарат) группы. При этом нахождение каждого из аппаратов группы задается алгоритмом, разработанным пользователем.

1.5. Использование камеры аппарата



При включении камеры происходит повышение нагрузки на CPU и GPU. Получаемый FPS зависит от технических характеристик ПК. Чтобы повысить FPS (в рамках ограничения), можно уменьшить размер окна Симулятора.



Чтобы включить камеру в окне Симулятора, нажмите на кнопку

Для получения изображений с камер аппарата используйте [ROS 2](#). Для получения изображений рекомендуется [написать подписчика](#) с использованием библиотеки [image_transport](#).

1.6. Горячие клавиши переключения видов

В Симуляторе доступны следующие виды:

- вид от первого лица (F1);
- вид от третьего лица (F2);
- вид со свободной камеры (F3);
- вид от лица пилота (F4);
- вид от третьего лица для группы моделей аппаратов (F5).

По клавише С происходит циклическое переключение между видами.

Переключение между аппаратами осуществляется:

- по номеру аппарата (нумерация начинается с 1) с помощью цифровых клавиш;
- на следующий и предыдущий аппараты с помощью клавиш X и Z соответственно.

1.7. Переключение на Vulkan

По умолчанию Симулятор использует OpenGL, но для повышения производительности имеется возможность переключиться на Vulkan.



Стабильная работа при переключении с OpenGL на Vulkan не гарантируется.

Для переключения на Vulkan в файле client/start.sh после первой строки добавьте строку:

```
GAPI="vk"
```

2. Начало работы

Запустите кластер симулятора:

```
./cluster.sh settings/ИМЯ.json
```

ИМЯ — имя файла параметров запуска, которое зависит от дисциплины и уровня сложности (один из json-файлов директории settings).



Устанавливая и запуская Симулятор, вы соглашаетесь с условиями лицензии.

3. Дисциплины

Интерфейс Симулятора позволяет тренироваться в следующих дисциплинах:

- [Синхронный полёт](#);
- [Командная гонка](#);
- [Уход от столкновения](#).

3.1. Синхронный полёт

В рамках дисциплины «Синхронный полет» задача — разработать и запрограммировать алгоритм, позволяющий группе из 6 беспилотных аппаратов преодолеть гоночную трассу, выстраиваясь перед трибунами зрителей в геометрические формации различной сложности.

3.1.1. Гоночная трасса

Гоночная трасса представляет собой футбольный стадион.

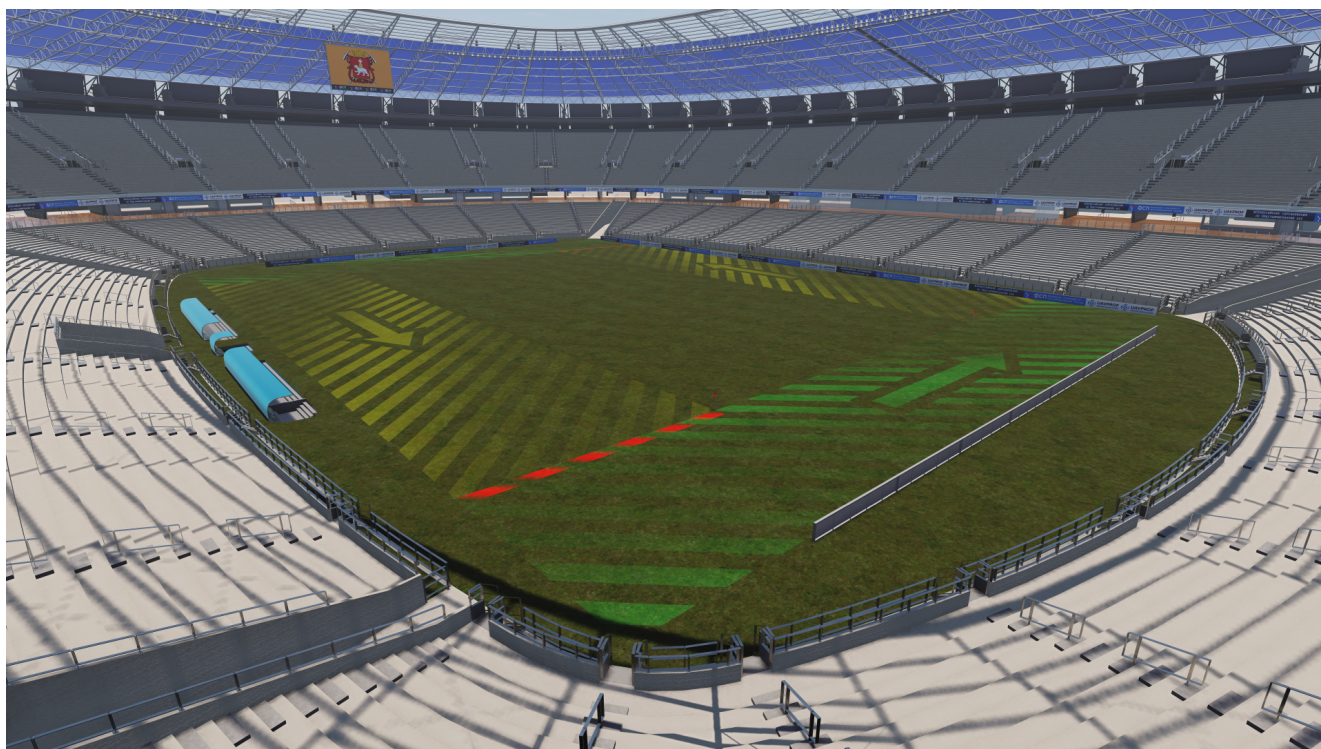


Рис. 1. Синхронный полёт. Трасса

Полет группы аппаратов осуществляется над выделенными зонами поля по направлению против часовой стрелки.

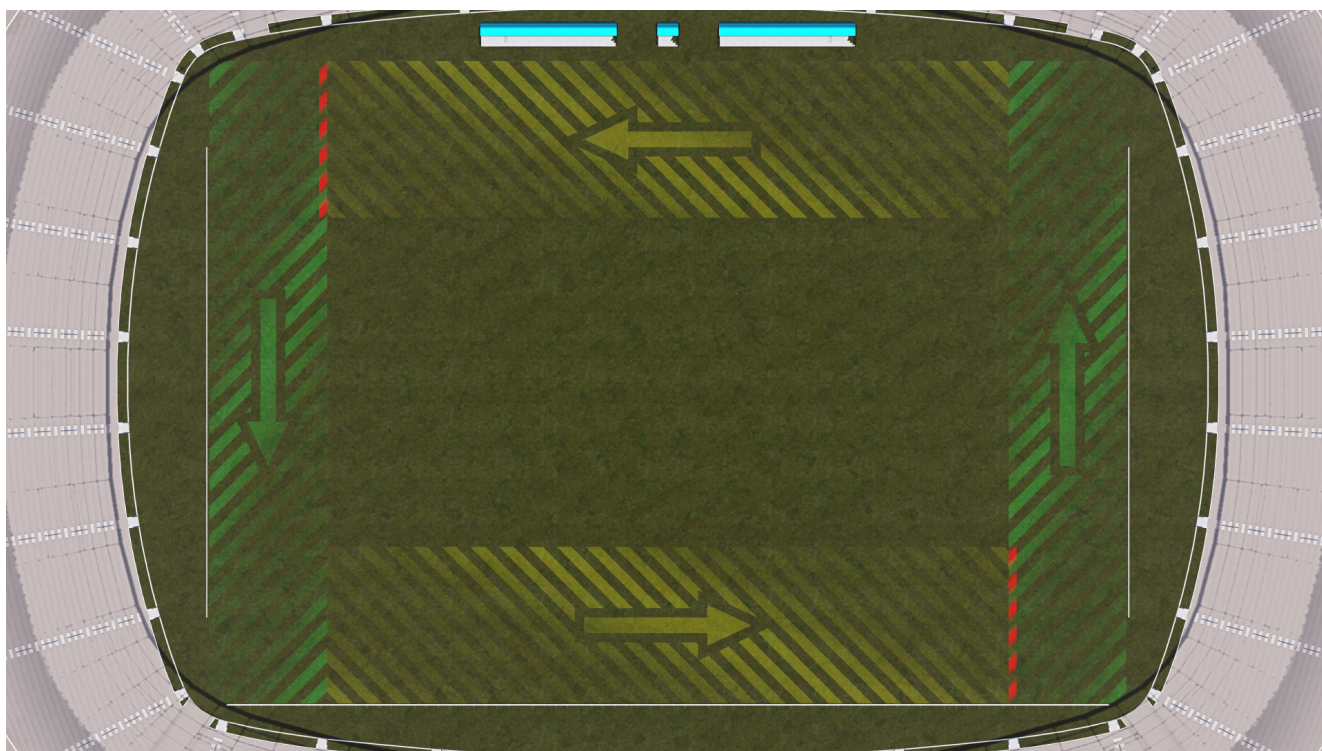


Рис. 2. Зоны поля

Зоны пролета — части поля, отмеченные желтым цветом.

Над зонами пролета группа аппаратов должна двигаться, поддерживая формацию (взаимное пространственное расположение) в соответствии с полученным заданием.

Размеры зон пролета, в метрах:

- длина — 87;
- ширина — 20;
- высота — 40.

Зоны перестроения — части поля, отмеченные зеленым цветом.

Пролетая над зонами перестроения, группа аппаратов должна перестроиться в новую формацию в соответствии с полученным заданием.

Размеры зон перестроения в метрах:

- длина — 82;
- ширина — 15;
- высота — 40.

Центры зон пролета расположены по следующим координатам: (0, 31) и (0, -31). Центры зон перестроения: (51, 0) и (-51, 0).

Место старта находится в первой зоне перестроения (51, 0).

3.1.2. Формации

Формации задаются в относительной системе координат (базис системы совпадает с базисом системы симулятора, а точка отсчета произвольна).

Например, построение в виде буквы «Т» из шести аппаратов может быть задано в таком виде:


```
(0, -3, 11), (0, -1, 11), (0, 1, 11), (0, 3, 11), (0, 0, 8), (0, 0, 5)
```

3.1.3. Задание

Задача пользователя — разработать алгоритм управления группой аппаратов, который автоматически:

- распределит месторасположение аппаратов в каждой из формаций;
- выдаст управляющие воздействия на аппараты для поддержания формаций;
- выдаст управляющие воздействия на аппараты для перестроения между формациями;
- выдаст управляющие воздействия на аппараты для перемещения по гоночной трассе для выполнения полного полетного задания.

Для Студенческого уровня

Полное полётное задание в виде последовательности и набора необходимых для выполнения формаций известно заранее:

- первая формация — буква «Т»:

```
(0, -3, 11), (0, -1, 11), (0, 1, 11), (0, 3, 11), (0, 0, 8), (0, 0, 5)
```

- вторая — «Е»:

```
(0, 0, 11), (0, 3, 11), (0, 0, 8), (0, 3, 8), (0, 0, 5), (0, 3, 5)
```

- третья — «С»:

```
(0, 1, 9), (0, 0, 11), (0, -2, 9), (0, -2, 7), (0, 0, 5), (0, 1, 6)
```

- четвертая — снова «Т».

В каждой зоне пролета (до выхода из нее) должна поддерживаться одна формация согласно полученному полетному заданию. После чего в зоне перестроения должно быть выполнено перестроение в следующую по полетному заданию формацию, и поддержание созданной формации в ближайшей зоне пролета по направлению движения (показано стрелками).

Для Профессионального уровня

При запуске Симулятора автоматически выдается полетное задание в виде необходимой для выполнения формации в ближайшей зоне пролета по направлению движения (показано стрелками). В тот момент, когда все аппараты переходят из зоны пролета в зону перестроения (красные линии), выдается следующее полетное задание в ближайшей зоне пролета по направлению движения.

Выдача полетного задания происходит с помощью системы ROS 2. При запуске симулятора также запускается ROS 2-нода, публикующая в ROS 2-топик `formation` текущие относительные координаты для построения формации в формате

```
geometry_msgs/msg/PoseArray
```

3.1.4. Запуск

Команда для запуска Симулятора выглядит следующим образом.

Студенческий уровень:

```
./cluster.sh settings/formation_stud_test_settings.json
```

Профессиональный уровень:

```
./cluster.sh settings/formation_prof_test_settings.json
```

3.1.5. Критерии оценивания

Измеряются следующие критерии:

- время прохождения трассы.

Таймер запускается при взлёте хотя бы одного аппарата, а завершается после прохождения зоны пролёта последней формации.

- Количество столкновений.

Повторное столкновение засчитывается только через 0,5 секунды после первого столкновения. Два столкновения учитываются единожды, если между ними прошло менее 0,5 секунды.

- Длительность вылета за пределы ограничительной зоны.

Суммарное время, проведённое как минимум одним аппаратом за пределами ограничительной зоны во время прохождения зоны пролёта.

- Отклонение от формации.

При пролёте зоны показа в каждый момент времени через равные промежутки времени измеряется среднеквадратическое отклонение (СКО) положения аппаратов от точек формации. Для каждой зоны показа вычисляется среднеквадратическое значение СКО (СКСКО). После этого вычисляется сумма СКСКО по зонам пролёта.

3.2. Командная гонка

В рамках дисциплины «Командная гонка» задача — разработать и продемонстрировать работоспособность алгоритма, позволяющего беспилотному аппарату пролететь гоночную трассу с использованием технического зрения.

3.2.1. Гоночная трасса

Гоночная трасса представляет собой последовательность из 7 ворот в помещении с размерами 25.94 x 39.19 x 8.41 м. Трасса является круговой: одни ворота являются стартовыми/финишными, а остальные — промежуточными. Положения и последовательность ворот не изменяются. Трасса находится в зоне размером 20 x 25 x 6 м.

Аппарату необходимо взлететь со взлётной площадки и пролететь 3 круга.

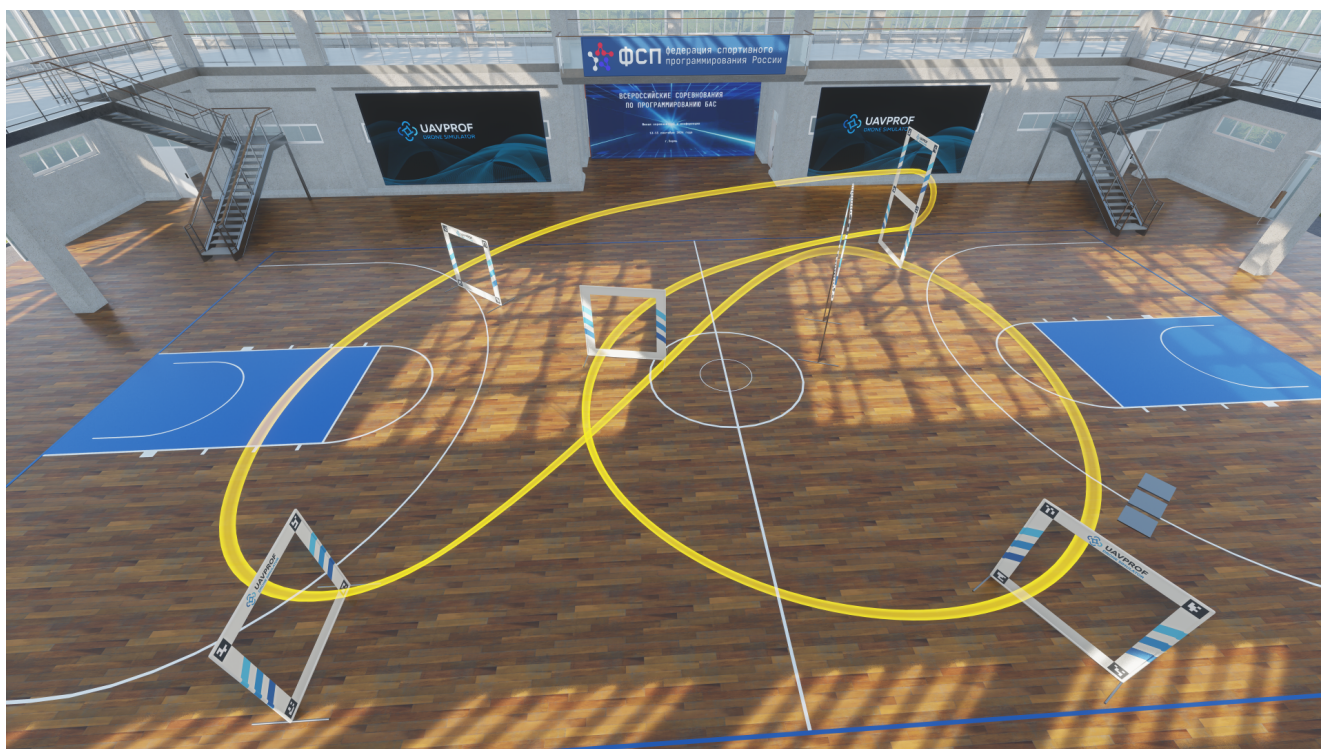


Рис. 3. Командная гонка. Трасса

Ворота — объект с квадратным отверстием 1.54 x 1.54 м, через которое необходимо пролететь аппарату. Чтобы пролететь ворота, аппарату необходимо центром масс пересечь плоскость отверстия (внутри отверстия) в правильном направлении.

На воротах со стороны, в которую нужно влететь, рядом с углами отверстия нанесены ArUco-маркеры размером 0.19 x 0.19 м (коды маркеров взяты из [словаря DICT_4X4_250 библиотеки OpenCV](#)). Номер маркера в словаре вычисляется по формуле:

$$M = N * 4 + C$$

где:

- $N \in [0..6]$ — последовательный номер ворот;
- $C \in [0..3]$ — номер угла:
 - **0** — левый нижний;
 - **1** — правый нижний;
 - **2** — правый верхний;
 - **3** — левый верхний.

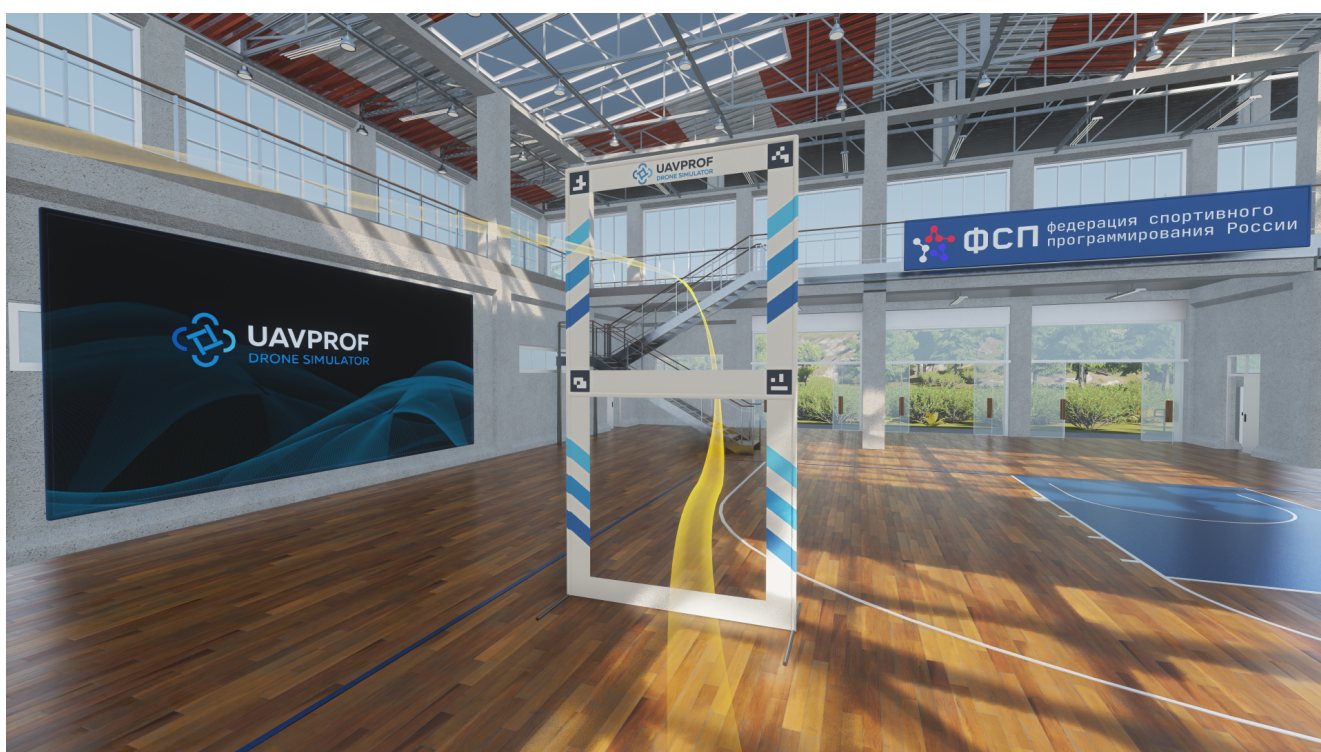


Рис. 4. Внешний вид ворот

Для визуализации (не присутствуют на изображениях с камеры аппарата) показаны: зона трассы (видна вблизи), линия трассы и отмечены ворота. Ворота, которые необходимо пролететь, окрашены в зелёный цвет. При нарушении направления полёта ворота окрашиваются в красный цвет. Следующие ворота — жёлтые, предыдущие — красные.

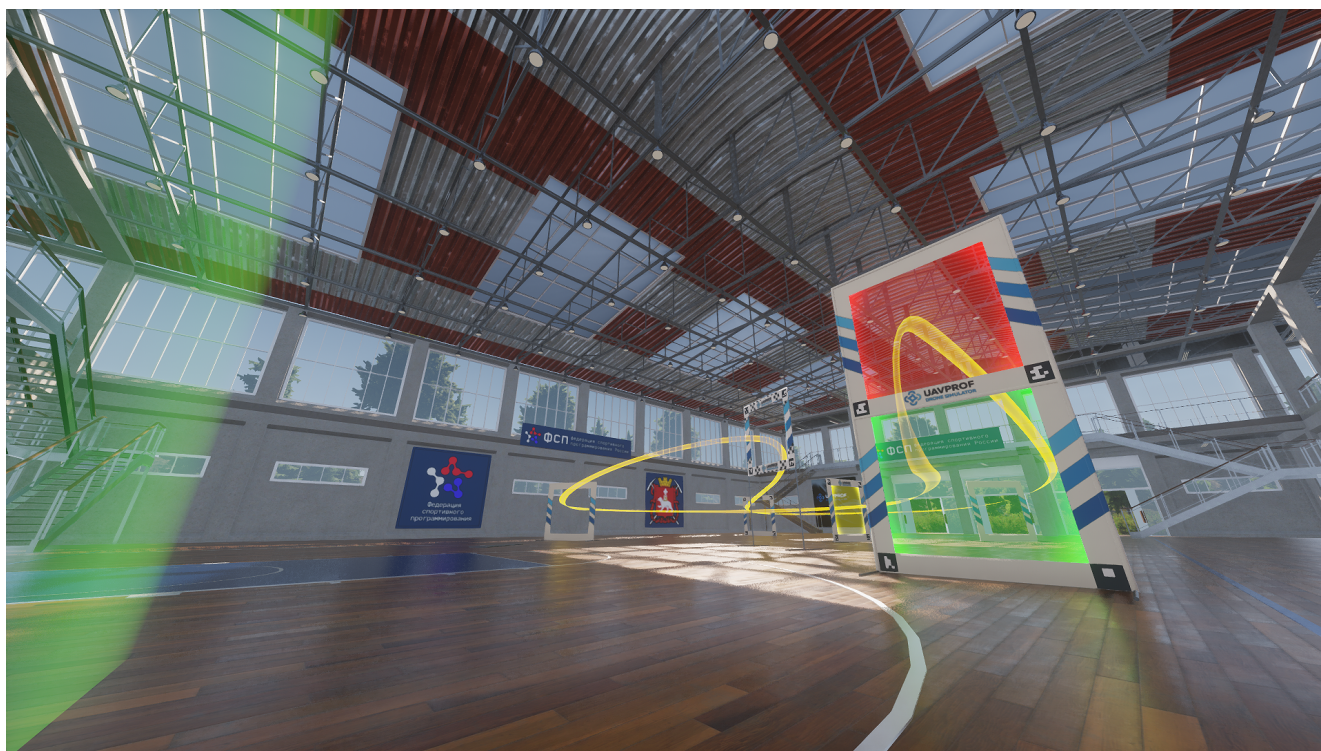


Рис. 5. Внешний вид ворот при пролёте через них

Все объекты трассы являются непроходимой преградой для аппаратов — на них работает

механизм упругого столкновения. В большинстве случаев это правило действует и для помещения (вне зоны трассы могут встречаться невидимые преграды).

3.2.2. Аппарат

Модель аппарата зависит от уровня сложности:

- для Студенческого — FPV F5D Auton LPS;
- для Профессионального — FPV F5D Auton.

3.2.3. Задание

Задача аппарата — взлететь и пролететь 3 круга гоночной трассы.

Задача пользователей — разработать алгоритм управления аппаратом, который автоматически выдаст управляющие воздействия на аппарат для выполнения задачи аппарата.

3.2.4. Запуск

Команда для запуска Симулятора выглядит следующим образом.

Студенческий уровень:

```
./cluster.sh settings/race_stud_test_settings.json
```

Профессиональный уровень:

```
./cluster.sh settings/race_prof_test_settings.json
```

3.2.5. Критерии оценивания

Измеряются следующие критерии:

- время прохождения трассы.

Таймер запускается при взлёте аппарата, а завершается после прохождения последних ворот.

- Количество столкновений.

Повторное столкновение засчитывается только через 0,5 секунды после первого столкновения. Два столкновения учитываются единожды, если между ними прошло менее 0,5 секунды.

- Длительность вылета за пределы ограничительной зоны.

Суммарное время, проведённое аппаратом за пределами ограничительной зоны.

3.3. Уход от столкновения

В рамках дисциплины «Уход от столкновения» задача пользователя — реализовать алгоритм управления, позволяющий БПЛА пролететь заданный маршрут, избежав столкновений с

препятствиями (воздушным шаром / птицами).

3.3.1. Гоночная трасса

Полигон представляет собой участок горной пересеченной местности.



Рис. 6. Уход от столкновения. Трасса

Маршрут представляет собой ломаную линию, заданную координатами ее вершин (точек маршрута). Полет по маршруту начинается при взлете аппарата и заканчивается при его посадке на посадочную площадку с расстоянием 3 метра до центра этой площадки. При полете по маршруту требуется строгое поддержание траектории. Допустимая зона находится вокруг линии маршрута не дальше некоторого расстояния **R** от нее.

В качестве препятствий выступают воздушный шар / птицы, местоположение которых не зависит от перемещений аппарата.

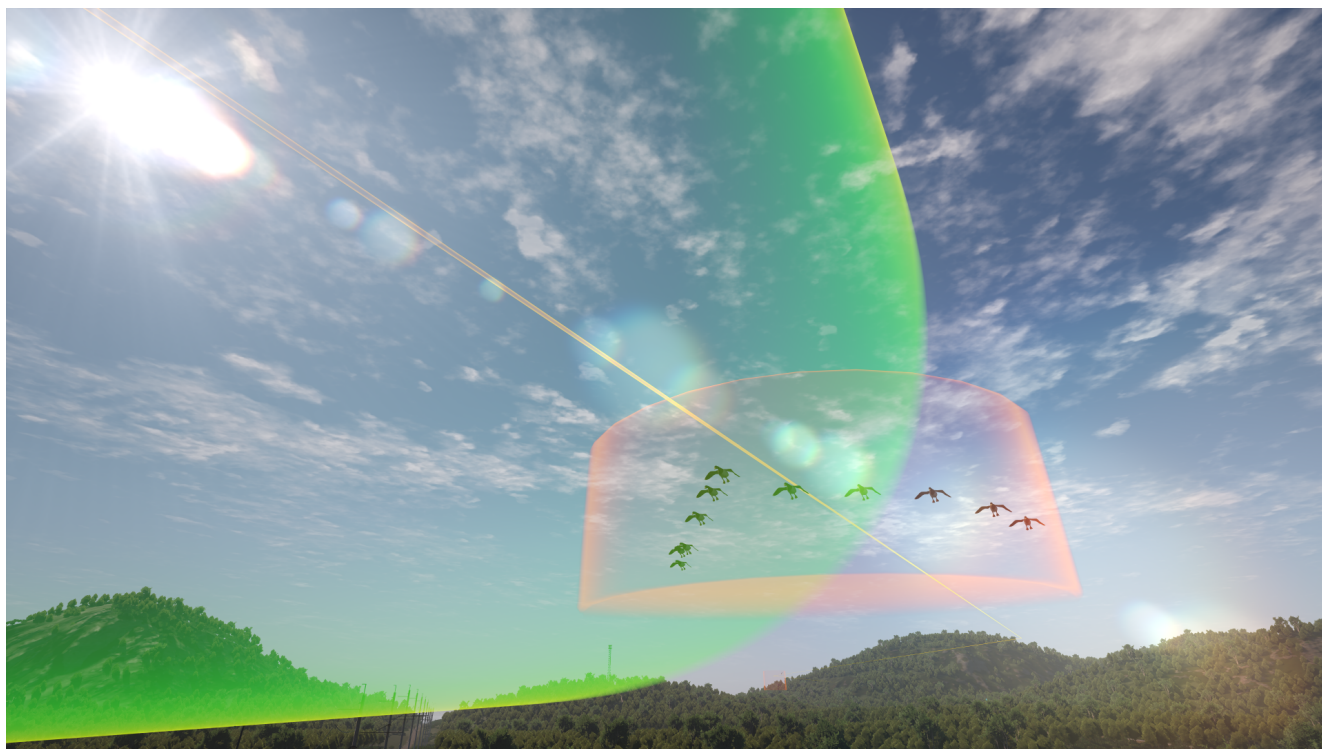


Рис. 7. Пример препятствия

Птицы имеют внешний коллайдер (объект, определяющий форму и размер зоны столкновения). Птиц следует облетать:

- на расстоянии 2-3 метра выше их траектории полета;
- 3-4 метра ниже их траектории;
- 5 метров слева/справа.

Воздушный шар обладает коллайдером, соответствующим его визуальной модели; его целесообразно облетать слева или справа.

3.3.2. Задание

Задача пользователя — разработать алгоритм управления аппаратом, который автоматически:

- выдаст управляющее воздействие на аппарат для перемещения по маршруту;
- определит препятствия, с которыми может произойти столкновение;
- выдаст управляющее воздействие для избегания столкновений.

Для Студенческого уровня положение препятствий, а также их габаритные размеры выдаются в течение полета аппарата, при нахождении препятствия на расстоянии обнаружения L от аппарата.

Для Профессионального уровня параметры динамических препятствий необходимо определять с помощью оптических камер аппарата.

Координаты точек маршрута в локальной системе координат Симулятора:

```
[ 549.75,    207.8,    98.56658],
[ 549.75,    207.8,    125.0    ],
[ 580.0,     366.0,    125.0    ],
[ 624.55501, 598.40931, 70.0    ],
```

```
[ 928.84280, 947.73771, 95.0 ],
[ 965.0, 1238.0, 100.0 ],
[1108.28750, 1443.92409, 105.0 ],
[ 933.8, 1701.4, 140.0 ],
[ 933.8, 1701.4, 107.66052]
```

Допустимое расстояние **R** вокруг линии маршрута, в метрах: 7.5

Расстояние обнаружения препятствий (для Студенческого уровня) **L**, в метрах: 10

Выдача информации о препятствиях (для Студенческого уровня) происходит с помощью системы ROS 2. При запуске симулятора также запускается ROS 2-нода, публикующая в ROS 2-топик `obstacles` информацию о препятствии в формате строки

```
std_msg/msg/String
```

Формат строки, публикуемой в топик; значения разделены пробелами:

```
obj_name x y z scale_x scale_y scale_z
```

Значения в строке:

- **obj_name** — имя препятствия;
- **x y z** — координаты препятствия;
- **scale_x scale_y scale_z** — габариты препятствия (в метрах).

Возможные значения **obj_name**:

- **hot_air_balloon** — воздушный шар;
- **eagles** и **ducks** — группы птиц.

3.3.3. Запуск

Команда для запуска Симулятора выглядит следующим образом.

Студенческий уровень:

```
./cluster.sh settings/avoid_stud_test_settings.json
```

Профессиональный уровень:

```
./cluster.sh settings/avoid_prof_test_settings.json
```

3.3.4. Критерии оценивания

Измеряются следующие критерии:

- время прохождения трассы.

Таймер запускается при взлёте аппарата, а завершается при приземлении аппарата на

посадочную площадку.

- Количество столкновений.

Повторное столкновение засчитывается только через 0,5 секунды после первого столкновения. Два столкновения учитываются единожды, если между ними прошло менее 0,5 секунды.

- Длительность вылета за пределы ограничительной зоны.

Суммарное время, проведённое аппаратом за пределами ограничительной зоны.

- Отклонение от маршрута.

При пролёте маршрута в каждый момент времени через равные промежутки времени измеряется отклонение аппарата от ближайшей точки на ломаной маршрута. После преодоления маршрута вычисляется среднеквадратическое отклонение (СКО) по всем измеренным отклонениям.

4. Работа со светодиодной панелью

Для работы со светодиодной панелью установите ROS 2 (рекомендуема версия — Humble). Установить ROS 2 и настроить окружение можно по ссылке: <https://docs.ros.org/en/humble/Installation.html>.

Для запуска симулятора с контейнерами соберите пользовательский контейнер с ROS 2.

4.1. API модуля



Рекомендуется работать со светодиодной матрицей в рамках одного запуска скрипта, так как для успешной отправки сообщений об изменении цвета светодиода должно установиться подключение.

Для работы со светодиодной матрицей импортируйте PioneerLed в скрипт:

```
from pioneer_led.pioneer import PioneerLed
```

После этого создайте экземпляр PioneerLed.

Можно включить logger — тогда в консоль будут выводиться индексы с цветом только что измененных светодиодов. Значение по умолчанию: logger=False:

```
pioneer = PioneerLed()
```

После этого изменять цвет светодиодов можно через функцию led_control:

```
pioneer.led_control(led_id, r, g, b)
```

- **led_id** (int, [0 .. 24, 255]) — номер светодиода (255 - все светодиоды);
- **r** (int, [0 .. 255]) — значение красного;
- **g** (int, [0 .. 255]) — значение зеленого;
- **b** (int, [0 .. 255]) — значение синего.

Таким образом можно обратиться к любому светодиоду (или ко всем сразу) на матрице и изменить его цвет.



Чтобы светодиод засветился, хотя бы одно из значений **r**, **g**, **b** должно быть больше **80**.

4.2. Разбор примера

Рассмотрим пример кода, который выводит на матрице крест:

```
from pioneer_led.pioneer import PioneerLed
```

```
import sys

leds = [1, 5, 7, 9, 13, 17, 19, 21, 25]

if __name__ == "__main__":
    try:
        pioneer = PioneerLed()
        pioneer.led_control(255, 0, 0, 0)

        color = [int(i) for i in sys.argv[1:4]]

        for led_id in leds:
            pioneer.led_control(led_id-1, *color)

    except Exception as e:
        print(e)
```

Создаём список номеров светодиодов, которые необходимо включить. Для удобства записываем номера, как они подписаны на плате, начиная с 1:

```
leds = [1, 5, 7, 9, 13, 17, 19, 21, 25]
```

Создаём экземпляр класса PioneerLed и выключаем все светодиоды (чтобы не выводить изображение поверх предыдущего).

```
pioneer = PioneerLed()
pioneer.led_control(255, 0, 0, 0)
```

Считываем цвет из аргументов командной строки.

```
color = [int(i) for i in sys.argv[1:4]]
```

В цикле включаем светодиоды из списка. Так как записывали номера, начиная с 1, необходимо вычесть 1.

```
for led_id in leds:
    pioneer.led_control(led_id-1, *color)
```

4.3. Запуск

1. Запустите симулятор:

```
./cluster.sh settings/led_settings.json
```

2. В отдельном терминале войдите в окружение контейнера центра управления (сеть межбортового обмена):

```
./exec_cc.sh bash -l
```

3. Подключите окружение ROS 2:

```
source /opt/ros/humble/setup.sh
```

4. Выставьте ROS_DOMAIN_ID сети межбортового обмена:

```
export ROS_DOMAIN_ID=100
```

5. Запустите скрипт при запущенном Симуляторе:

```
python3 x_tutorial.py 255 0 0
```

Отображение светодиодов на матрице:

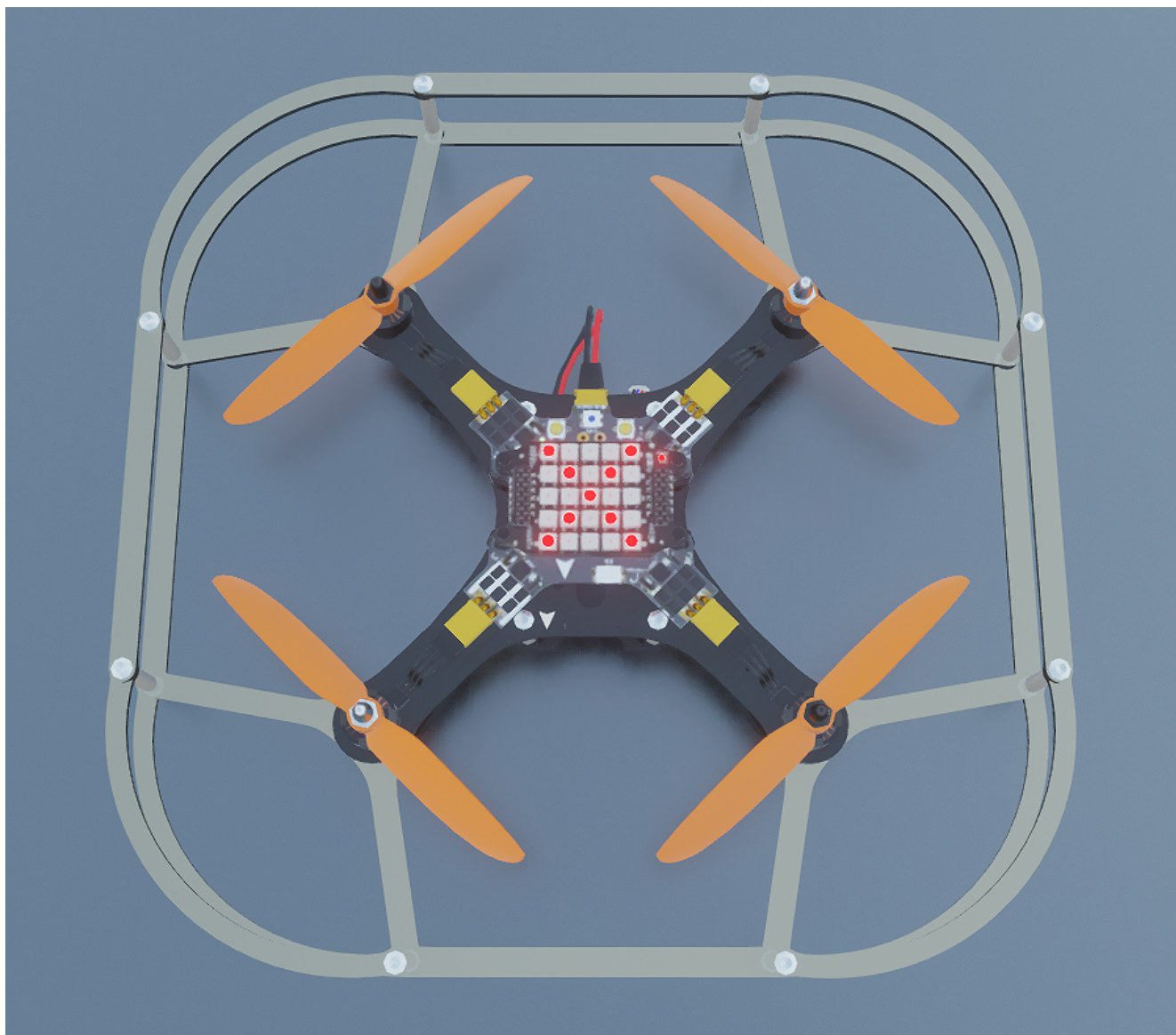


Рис. 8. Светодиоды на матрице, знак X

4.4. Дополнительные примеры

Примеры запускаются из директории, в которой находятся.

4.4.1. Управление одним светодиодом

Для запуска примера введите в терминале:

```
python3 led_control_one_diod.py led_id r g b
```

Значения в строке:

- **led_id** - `int[0 .. 24, 255]` — номер светодиода, к которому применяем смену цвета.

Если ввести **255**, то обращение будет ко всем светодиодам.

- **r, g, b - int[0 .. 255]** — значение цвета (красный, зеленый, синий соответственно).

После этого на матрице засветится тот светодиод, индекс которого был введен. Если нужно изменить цвета всей матрицы, то введенный индекс должен быть равен **255**.

4.4.2. Цвета

Для запуска примера введите в терминале:

```
python3 led_control_all_colors.py
```

В данном примере показаны возможные способы управления светодиодной матрицей. Можно задавать цвет как всей матрице одновременно, так и для каждого отдельного светодиода.



Рис. 9. Цвета

4.4.3. Цифры

Для запуска примера введите в терминале:

```
python3 led_print_numbers.py led_id r g b
```

Значения в строке:

- **r, g, b - int[0 .. 255]** — значение цвета (красный, зеленый, синий соответственно).

В данном примере показаны возможные способы использования `led_control`. При помощи

заранее заданных индексов можно рисовать любые символы/изображения в рамках матрицы размером 5x5.



Рис. 10. Цифры

4.4.4. Смайлики

Для запуска примера введите в терминале:

```
python3 led_print_smiles.py r g b
```

Значения в строке:

- **r, g, b - int[0 .. 255]** — значение цвета (красный, зеленый, синий соответственно).



Рис. 11. Смайлик

История изменений

15.11.2024

Документ создан.