Летающая робототехника

2022/23 учебный год

Инженерный тур

Общая информация

Разработка системы автоматизированного поиска очагов возгорания в помещении при помощи автономного квадрокоптера.

Сюжет задачи

Быстрое обнаружение и локализация возгораний — важнейшая задача экстренных служб. Квадрокоптеры успешно применяется пожарными и спасателями во всем мире. Современные пожарные команды успешно внедряют дроны, а также полезную нагрузку нового поколения.

Все это позволяет принимать обоснованные решения в условиях чрезвычайных ситуаций, а под тактикой применения квадрокоптеров для пожаротушения понимается поиск наиболее целесообразных способов и приемов борьбы с огнем в конкретных условиях. Кроме того, квадрокоптеры позволяют получать данные о местности и топографии, что помогает лучше планировать действия пожарных и спасателей.

Руководителю тушения пожара намного проще составить тактический план ликвидации огня на основании данных мониторинга как внутри помещения, так и снаружи.

В свою очередь, разведка с помощью квадрокоптеров должна установить:

- вид, силу и локацию пожара;
- наличие особо ценных участков и пожароопасных зон;
- наличие преград;
- безопасные места.

Квадрокоптеры также могут использоваться для поставки воды и других материалов на место возгорания, а также для контроля за процессом тушения пожара.

Польза от использования квадрокоптеров для пожаротушения очевидна и неоспорима. Они помогают пожарным спасать людей и имущество в максимально быстрое время, что позволяет снизить убытки и риски, связанные с пожарами.

Требования к команде и компетенциям участников

Количество участников в команде: 3–4 участника.

Роль 1. Инженер-программист (Python) — программирование на Python, C++, работа с SSH, ROS, разработка алгоритмов компьютерного зрения для реализации автономных миссий квадрокоптера.

Роль 2. Инженер-программист (C++, Python) — алгоритмы компьютерного зрения для реализации автономных миссий квадрокоптера, программирование на Python,

работа с SSH и ROS. Работа в связке с ролью 1.

Роль 3. Инженер-техник — моделирование функционального узла квадрокоптера, разработка документации включающей инструкции по сборке, установке и эксплуатации устройства, техобслуживание квадрокоптера, тестирование системы, пилотирование БПЛА, моделирование и изготовление функционального узла квадрокоптера.

Роль 4. Капитан/лидер команды — работа с системами построения карты в RViz, осуществление общего руководства работой команды, распределение обязанностей и контроль соблюдения дедлайнов. Рекомендуется совмещение данной роли с другими ролями.

Оборудование и программное обеспечение

Наименование	**Описание**
Полетный полигон ($5 \times 7 \times 3$ м)	Полигон для запуска автономных полетных миссий
Ноутбук (Ryzen 5 5500u 2.1ghz 20gb ram)	Разработки программного кода для запуска полетной миссии
Графическая станция (Ryzen 7 1700 3.7ghz 16gb ram Geforse gtx 1060 3gb)	Разработка полезной нагрузки
Конструктор программируемого квадрокоптера «COEX Клевер 4 Code»	Запуск полетных миссий
3D Принтер	Печать разработанных деталей для изготовления полезной нагрузки
• T-flex CAD 17 • Компас 3D v21 • Inventor 2022 • Cura • Repetier host	Программное обеспечение для разработки и печати полезной нагрузки
• OBS • VLC player	Программное обеспечение для записи выполнении задания в RViz

Наименование	Описание
• VMWare Workstation (Gazebo) • Putty • Winscp • Notepad++ • QgroundControl • Etcher • ColorMania • Arduino ide • PyScripter • VScode • Python 3 • Termius	Программное обеспечение для работы с программным кодом

Описание задачи

Программная часть

Начальные условия:

1. Зона «H» является точкой взлета и точкой посадки;
2. Конфигурация застройки может изменяться. Стены могут находиться только параллельно осям координат (с погрешностью 10°);
3. Границей зоны, в который могут находиться возгорания, является стена, помещение всегда находится со стороны севера (на рисунке VI.2.1);
4. Количество, положение и площадь возгораний может меняться;
5. Количество и положение пострадавших может меняться;
6. Положение датчиков на квадрокоптере может быть изменено при необходимости;
7. Миссия мониторинга должна проходить полностью в автономном режиме;
8. Начало координат Aruco карты в нижнем левом углу $(0, 0)$;
9. В случае подгона координат, визуализации, и т. д., организаторы оставляют за право, принять решение по оценке индивидуально;
10. Классы пожара:
    - Класс A: уголь (`coal`), текстиль (`textiles`), пластмасса (`plastics`).
    - Класс B: нефтепродукты (`oil`), спирт (`alcohol`), глицерин (`glycerine`).

Рис. VI.2.1. Расположение полигона

Для успешного выполнения задания участникам необходимо:

1. **Взлет**:
   - 1.1. Совершить автономный взлет с зоны «H».
   - 1.2. Установить значение высоты дрона не более 1,5 метров, возможно изменение в течение миссии, но только в меньшую сторону (не поднимать дрон выше стены).
2. **Автоматизированный поиск очагов возгорания в помещении при помощи автономного квадрокоптера**:
   - 2.1. Прилететь к точке начала мониторинга (постоянна, известна).
   - 2.2. Начать движение по любой траектории, позволяющей обнаружить возгорания, находящиеся внутри помещения (светло-серая зона на рис. VI.2.1).
     - 2.2.1. В случае выхода проекции квадрокоптера за пределы помещения на 5 секунд и более, считается, что дрон окончил мониторинг и дальнейших действий не планируется. Дальнейшие баллы не начисляются.
   - 2.3. Определить длину каждой стены, согласно нумерации ниже:

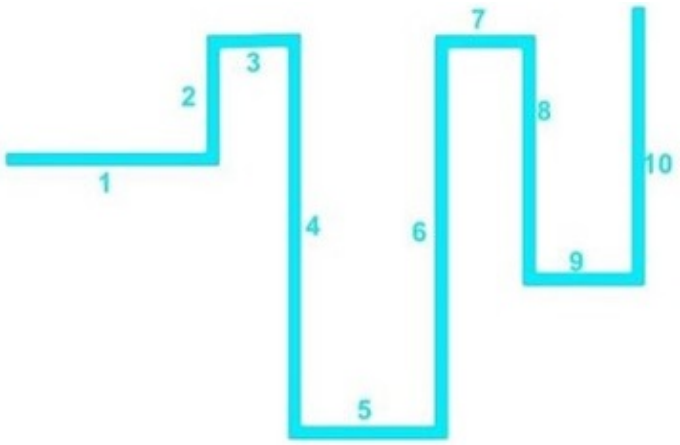

Рис. VI.2.2. Длины стен

2.3.1. Отобразить каждую стену, привязанную к системе координат, в системе визуализации любым возможным методом (`marker array`, `point cloud`, и т. д.; важно чтобы была возможность различить стены на экране).

2.4. Обнаружить все возгорания (при их наличии).

2.4.1. Отобразить возгорания, привязанные к системе координат в системе визуализации (`Rviz` или аналоге, используя `marker array`), форма и размер значение не имеет, цвет — оранжевый.

2.4.2. Сделать запрос на сервер с координатами возгорания. Формат запроса указан в OpenAPI документации (http://65.108.222.51/docs).

2.4.3. В зависимости от полученного результата определяем класс пожара (данная функция необходима для выполнения дальнейшего задания).

2.4.4. Вывести в терминал сообщение о координатах обнаруженных возгораний относительно `aruco_map` типе пожара и хранящемся в нем веществе. Формат: `fire: <x> <y> <material> <class>`
В пункте `<material>` указывается конкретный тип.

2.5. Обнаружить пострадавших в пожаре рабочих. На всех рабочих были надеты каски синего цвета.

2.5.1. Вывести в терминал координаты обнаруженных пострадавших Формат: `injured: <x> <y>`

2.5.2. Отобразить местонахождение пострадавших с привязкой к системе координат в системе визуализации (`Rviz` или аналоге, используя `marker array`), форма и размер значение не имеет, цвет — синий.

2.5.3. Определить ближайший очага возгорания.

2.6. Продолжить мониторинг, в случае обнаружения иных возгораний повторить пункт 2.3–2.5;

2.7. По окончанию мониторинга помещения, вернуться в зону «H»;

3. **Совершить автономную посадку в пределах зоны «H».**
4. **Сгенерировать автоматический отчет и вывести в терминал сообщение в виде:**

Общий вид формата вывода	Пример вывода

`Fires: <количество возгораний>`
 	
`Fire 1: <x> <y> <material> <class>`
`Fire 2: <x> <y> <material> <class>`
`Fire ...: <x> <y> <material> <class>`
 	
`Injured: <количество пострадавших>`
 	
`Injured 1: <x> <y> <nearest_fire>`
`Injured 2: <x> <y> <nearest_fire>`
`Injured ...: <x> <y> <nearest_fire>`
 	
`Wall 1: <length>`
`Wall 2: <length>`
`Wall ...: <length>`
`Wall 10: <length>`
	`Fires: 3`
	
	`Fire 1: 1.0 1.4 coal A`
	`Fire 2: 5.0 3.0 oil B`
	`Fire 3: 7.9 8.5 oil B`
	`Injured 1: 0 0 1`
	`Injured 2: 5.6 2.7 2`
	`Injured 3: 7.8 9.2 3`
	
	`Wall 1: 2.0`
	`Wall 2: 0.5`
	`Wall 3: 1.0`
	`Wall 4: 3.0`
	`Wall 5: 1.55`
	`Wall 6: 0.55`
	`Wall 7: 1.2`
	`Wall 8: 2.3`
	`Wall 9: 1.9`
	`Wall 10: 2.1`

Где x и y — координаты в метрах	

Инженерная часть

Техническое задание

на разработку функционального устройства для программируемого квадрокоптера «COEX Клевер 4 Code», с целью расширения его функциональных возможностей.

1. **Наименование выполняемых работ**
   Разработка дополнительного, модульного устройства для автономного тушения возгораний различного класса.
2. **Цель выполнения работ**
   Расширение функциональных возможностей программируемого квадрокоптера «COEX Клевер 4 Code», для возможности осуществления автономного тушения возгорания различного класса.
   - Класс A — пожары твердых горючих веществ и материалов;
   - Класс B — пожары горючих жидкостей или плавящихся твердых веществ и материалов.
3. **Срок выполнения работ**
   2 рабочих дня
4. **Минимальные требования к устройству**:
   - характеристика одной капсулы для пожаротушения класса «A»: высота 3,5 см, диаметр 1,5 см, форма цилиндрическая. Всего: 4 шт;
   - характеристика одной капсулы для пожаротушения класса «B»: кубическая форма со стороной 2,5 см. Всего: 4 шт;
   - модульность;
   - возможность изготовления посредством 3D печати (FDM принтер);
   - возможность осуществления сборки устройства посредством винтового/шпоночного соединения;

- возможность быстрого монтажа на квадрокоптер и демонтажа соответственно;
- максимальный вес устройства 200 г. (без учета веса капсул);
- возможность реализации автономного сброса капсул для тушения возгораний определенного типа;
- совместимость с программируемым квадрокоптером «COEX Клевер 4 Code»;
- возможность надежного монтажа на квадрокоптер.

5. **Состав работ**:
   Разработка 3D модели функционального устройства, удовлетворяющую минимальным требованиям (п.4.):
   5.1. 3D модель функционального устройства в сборке.
   5.2. 3D модель функционального устройства в разобранном виде (каждый компонент отдельно).
   5.3. Инструкция по сборке, монтажу и эксплуатации функционального устройства:
   - Спецификация (Гост 2.109-73: https://drive.google.com/file/d/1BKR930fFXkLQBRlPpwxInyyJr9_Fn1Dn/view):
     - наличие всех сборочных единиц;
     - наличие всех деталей;
     - наличие стандартных изделий.
   - Сборочный чертеж:
     - указаны сборочные единицы, детали и стандартные изделия с указанными взаимосвязями различных частей.
   - Монтажная инструкция:
     - наличие изображения монтируемого изделия;
     - наличие изображения всех изделий, применяемых при монтаже;
     - наличие перечня всех составных частей, необходимых для монтажа;
     - технические требования к монтажу изделия;
     - наличие электромонтажного чертежа;
     - наличие описания последовательности действий для осуществления монтажа изделия.
   - Инструкция по эксплуатации — указания, необходимые для правильной и безопасной эксплуатации устройства: этап предполетной подготовки, в процессе выполнения миссии и по завершению миссии.

6. **Результат работы**
   Устройство для возможности осуществления автономного тушения возгораний различного класса при помощи программируемого квадрокоптера «COEX Клевер 4 Code», соответствующее минимально заданным требованиям (п. 4 Технического задания).
7. **Сдача работы**

№ п.п.	Наименование	Формат
1.	3D модель функционального устройства в сборке	`.step` и `.stl` и проприетарный формат
2.	3D модель функционального устройства разобранном виде, каждый компонент отдельно	`.step` и `.stl`
3.	Инструкция по сборке, монтажу и эксплуатации функционального устройства	`.pdf`

Оценка результатов происходит только в тех случаях, если все составляющие итогового результата:

- предоставлены организаторам не позднее указанного срока;
- все материалы технической документации являются описанием устройства, разработанного командой согласно техническому заданию (3D модель функционального устройства в сборке; 3D модель функционального устройства разобранном виде, каждый компонент отдельно, инструкция по сборке, монтажу и эксплуатации функционального устройства).

Оценка работоспособности разработанного устройства происходит в процессе его эксплуатации.

Система оценивания

Во время инженерного тура участники получают баллы за выполнение зачетных миссий согласно критериям оценки успешности прохождения миссии.

Оценка инженерного задания разделена на 2 части:

- оценка работоспособности функционального узла квадрокоптера во время прохождения автономной миссии;
- оценка технической документации функционального узла квадрокоптера.

В таблице представлены критерии оценки заключительной зачетной миссии.

№ п/п	Критерий (Условия выполнения миссии)	Баллы за один случай	Кол-во случаев	Баллы за все случаи
1.	Автоматизированный поиск очагов возгорания в помещении *(оценивается во время полета)*			
1.1.	Длина стены определена верно (±0,2 м)	0,5	10	5
1.2.	Стена корректно выведена в визуализаторе (на нужном месте, имеет правильную длину; в случае использования Marker, высота не имеет значения)	0,5	10	5
1.3.	Координаты возгорания определены верно (±0,15 м)	0,9	4	3,6
1.4.	Запрос на сервер совершен, горящий материал и его класс определены верно	0,2	4	0,8
1.5.	Возгорание корректно выведено в визуализаторе (на нужном месте; в случае использования Marker — размер в разумных пределах не имеет значения)	0,3	4	1,2
2.	Обнаружение пострадавших *(оценивается во время полета)*			
2.1.	Координаты нахождения пострадавших определены верно (±0,15 м)	1,5	3	4,5

№ п/п	Критерий (Условия выполнения миссии)	Баллы за один случай	Кол-во случаев	Баллы за все случаи
2.2.	Пострадавший корректно выведен в визуализаторе (на нужном месте; в случае использования Marker — размер в разумных пределах не имеет значения)	0,3	3	0,9
2.3.	Ближайший очаг возгорания определен верно	0,1	3	0,3
3.	Визуализация *(оценивается во время полета)*			
3.1.	Открыт rviz, на котором выведено положение коптера, начала фреймов `aruco_map`, map любым способом.	0,5	1	0,5
3.2.	В rviz присутствует топик для вывода стен	0,5	1	0,5
3.3.	В rviz присутствует топик для вывода возгораний и пострадавших	0,5	1	0,5
4.	Посадка *(оценивается во время полета в случае, если выполнен как минимум один подпункт из первого блока (1.1–1.5))*			
4.1.	Посадка в пределах зоны «H»	1	1	1
5.	Автоматический отчет *(оценивается после выполнения зачетной миссии)*			
5.1.	Отчет записан корректно, не содержит ошибок (см. пример)	0,2	1	0,2
6.	Оценка результата *(оценивается после выполнения зачетной миссии)*			
6.1.	Код зачетной попытки с комментариями выложен на Github	0,5	2	1
		Итого за решение задач		**25**
7.	Штрафы *(оценивается после выполнения зачетной миссии)*			
7.1.	*Оценивается только при выполнении хотя бы одного случая 1.1.* В программе отсутствует логика считывания данных с датчика (любого) и выполняющая подсчет длин стен и зоны помещения (зоны, где находятся возгорания)	−5	1	−5
7.2.	*Оценивается только при превышении количества возгораний и пострадавших реального (п. 1.3, 2.1) в отчете* Обнаружены более 2 лишних очагов возгорания или пострадавших (за каждый случай)	−1	за каждый случай	

В таблице представлены критерии оценки выполнения технического задания.

№ п/п	Критерии оценки технической документации	Кол-во случаев	Баллы за один случай	Баллы за все случаи
1.	3D модель функционального устройства в сборке			
1.1	Оригинальное инженерное решение	2	0,5	1
1.2	Наличие всех элементов сборки	1	0,5	0,5
2.	3D модель функционального устройства в разобранном виде			
2.2	Наличие всех составных деталей сборки с соответствие с разработанной 3D моделью	1	0,5	0,5
3.	Инструкция по сборке, монтажу и эксплуатации			
3.1.	Спецификация *(соответствует разработанному устройству)*			
3.1.1	В спецификации правильно указаны все сборочные единицы	1	0,5	0,5
3.1.2	В спецификации правильно указаны все детали	1	0,5	0,5
3.1.3	В спецификации правильно указаны все стандартные изделия	1	0,5	0,5
3.1.4	В спецификации правильно указаны дополнительные обозначения	1	0,5	0,5

№ п/п	Критерии оценки технической документации	Кол-во случаев	Баллы за один случай	Баллы за все случаи
3.2.	Сборочный чертеж *(соответствует указанным в п. 5.3. требованиям и разработанному устройству)*			
3.2.1.	Раскрыты (показаны) все взаимосвязи деталей	3	0,25	0,75
3.2.2	Номера позиций указаны верно	1	0,25	0,25
3.2.3	Указаны все габаритные размеры и все размеры присоединения	1	0,75	0,75
3.3.	Монтажная инструкция *(соответствует указанным в п. 5.3. требованиям и разработанному устройству)*			
3.3.1.	Наличие списка всех необходимых крепежных изделий для монтажа	1	0,5	0,5
3.3.2.	Наличие на чертеже указания места монтажа разработанного устройства на квадрокоптере, крепежа и наглядного изображения монтажа	2	0,5	1
3.3.3	Наличие списка операций, которые необходимо произвести для установки изделия на квадрокоптер	2	0,25	0,5
3.3.4	Наличие электромонтажного чертежа	1	0,5	0,5
3.4.	Инструкция по эксплуатации *(соответствует разработанному решению)*			
3.4.1.	Наличие основных указаний, необходимых для правильной и безопасной эксплуатации, а также обслуживания устройства: этап предполетной подготовки, в процессе выполнения миссии и по завершению миссии. *(программный код, при необходимости его применения во время эксплуатации устройства)*	3	0,25	0,75
			Итого за решение задач	**9**

Программная часть

В первую очередь стоит отметить, что на очном финале, в отличие от распределенного, участникам предстоит не только работать с программированием квадрокоптера, но и с его настройкой и работой с возможными неполадками. Большая часть информации об этом доступна на сайте https://clover.coex.tech/, однако участники должны иметь сформированный опыт работы с оборудованием на PX4, Raspberry Pi и понимание как гуглить и решать те или иные ошибки.

Непосредственно решение задачи начнем с наиболее сложной ее части — детектирование, измерение и визуализация стен. Сначала стены необходимо детектировать. Здесь перед участниками стояла задача рассмотреть возможные способы работы с информацией о стенах. Были возможны следующие варианты:

- Использование камеры для детектирования стен *(более сложная работа с кодом, однако больше всего возможностей из-за fisheye камеры)*.
- Снятие дальномера с нижней деки квадрокоптера и установка на переднюю/боковую часть *(необходимость работы инженера, 3D принтера; возможное ухудшение качества полета)*.
- Установка еще одного дальномера *(сложность работы с двумя дальномерами на Raspberry Pi, необходимость работы инженера, 3D принтера)*.

Более подробно опишем первый вариант.

Сначала необходимо выполнить предобработку кадра с камеры.

```
def on_frame(self, img: np.ndarray, mask_floor: np.ndarray, hsv: Optional[np.ndarray]
↪  = None) -> None:
    """
    Функция обработки нового кадра с камеры.
    """
    img_0 = cv2.resize(img, (160, 120))
    t1 = time.time()
    if hsv is None:
        hsv_0 = cv2.cvtColor(img_0, cv2.COLOR_BGR2HSV)
    else:
        hsv_0 = cv2.resize(hsv, (160, 120))
    if self.cm is None:
        return None
    debug = img_0.copy()

    # получение маски границ стенок

    mask_floor_0 = cv2.resize(mask_floor, (160, 120))
    hsv_mask = cv2.inRange(hsv_0, self.mask[0], self.mask[1])

    mask = hsv_mask.copy()
    mask = cv2.erode(mask, np.ones((self.erode_kernel_size,
    ↪  self.erode_kernel_size),np.uint8), iterations=1)
    mask = cv2.morphologyEx(mask, cv2.MORPH_OPEN, np.ones((self.opening_kernel_size,
    ↪  self.opening_kernel_size),np.uint8), iterations=self.opening_iterations)
    mask = cv2.dilate(mask, np.ones((self.erode_kernel_size,
    ↪  self.erode_kernel_size),np.uint8), iterations=1)

    canny = cv2.Canny(mask,100,200)

    canny_masked = cv2.bitwise_and(canny, canny, mask=self.legs_mask)

    # точки границ стенок
    points = np.array(np.where(canny_masked))[::-1, :].astype(np.float64).T
    for i in points:
        cv2.circle(debug,(int(i[0]),int(i[1])), 1, (0,0,255), -1)

    if len(points) > 0:
        # расчет лучей из камеры проходящих через точки на кадре
        pnt_img_undist = cv2.undistortPoints(points.reshape(-1, 1, 2), self.cm,
        ↪  self.dc, None, None).reshape(-1, 2).T
        ray_v = np.ones((3, pnt_img_undist.shape[1]))
        ray_v[:2, :] = pnt_img_undist
        ray_v /= np.linalg.norm(ray_v, axis=0)

        # получение преобразования координат из main_camera_optical в aruco_map
        try:
            transform = self.tf_buffer.lookup_transform("aruco_map",
            ↪  "main_camera_optical", rospy.Time())
        except tf2_ros.ConnectivityException:
            print("LookupException")
            return None
        except tf2_ros.LookupException:
            print("LookupException")
            return None
        R_wb = np.array(quaternion_matrix([transform.transform.rotation.x,
        ↪  transform.transform.rotation.y, transform.transform.rotation.z,
        ↪  transform.transform.rotation.w]))[:3, :3]
        t_wb = np.array([transform.transform.translation.x,
        ↪  transform.transform.translation.y, transform.transform.translation.z])
```

```
        # применение преобразования лучам
        ray_v = np.matmul(R_wb, ray_v).T
        ray_o = t_wb

        # расчет координат границ стен в плоскости пола
        pnts = [intersect_ray_plane(v, ray_o) for v in ray_v]
        pnts = [p for p in pnts if p is not None]
        self.on_pnts(pnts)
```

На выходе имеем точки границ стен в плоскости пола, над которыми уже можем проводить операции по поиску стен специальным алгоритмом.

Алгоритм работает со списком стен, состоящих из отрезков и их состояний. Стены могут быть фиксированными, обновляемыми или неиспользуемыми. Для обработки новых отрезков (hls) используются следующие функции:

- `linesp_agg1` — построение среднего отрезка из списка отрезков путем нахождения среднего вектора и средней точки, а также определения новых конечных точек отрезка.
- `distance_2` и `simple_distance` — функции расстояния между отрезками.
- `rotated_rect_from_line` и `rect_overlap_metric` —- функции для вычисления повернутых прямоугольников из отрезков и метрики схожести между ними.
- `assign_to_prev` — сопоставление новых отрезков со списком предыдущих отрезков с использованием заданной функции расстояния и порогового значения.
- `create_new_pls_dbscan` — кластеризация новых отрезков с использованием алгоритма DBSCAN и заданных параметров.
- `update_pls` — обновление списка предыдущих отрезков с использованием новых отрезков и индексов соответствия.

Основной класс `WallsProcessor` инициализируется с заданными параметрами и использует вышеуказанные функции для обработки новых отрезков. Во время обработки алгоритм выполняет следующие действия:

- Сопоставляет новые отрезки (`hls_clustered`) с текущим списком стен (`pls`).
- Обновляет стены, если они сопоставлены с новыми отрезками и имеют состояние обновляемые.
- Добавляет новые отрезки в список стен, если они не были сопоставлены и находятся на достаточном расстоянии от текущей стены. Также учитывается угол между новым отрезком и текущей стеной.

На вход алгоритму поступают предобработанные отрезки, которые получаются из точек границ стен в плоскости пола следующим образом.

```
def on_pnts(self, pnts: List[np.ndarray]):
    """
    Обработка новых точек (добавление в карту)
    """
    xy = np.array(pnts)[:, :2]
    # print("xy:", xy)
    ijs = np.round((xy.reshape(-1, 2)[:, :2] - self.map_origin) /
    ↪  self.resolution).astype(np.int)[:, ::-1]
    ijs = ijs[(ijs[:, 0]>=0) & (ijs[:, 1]>=0)& (ijs[:, 1] < self.map_a.shape[1])&
    ↪  (ijs[:, 0] < self.map_a.shape[0])]
    # print("ijs:", ijs)
    self.map_a[ijs[:, 0], ijs[:, 1]] = np.clip(self.map_a[ijs[:, 0], ijs[:, 1]] + 10,
    ↪  0, 244)
```

```
        # self.map_a[ij[0], ij[1]] = 255

def proc_map(self):
    """
    Обработка карты.
    Поиск и фильтрация линий (стен).
    """
    debug_map = np.zeros((*self.map_a.shape, 3), dtype="uint8")
    map_a_pr = self.map_a.copy()

    self.map_a = map_a_pr

    # детектор линий probabilistic Hough transform algorithm
    linesP = cv2.HoughLinesP(255*(self.map_a > 100).astype("uint8"), 1, np.pi / 200,
    ↪  15, None, 10//3, 50//3)

    ress = []
    if linesP is not None:
        hls = [LineSP.from_2pnt(*[self.map_ij2xy([l[0][1], l[0][0]]),
        ↪  self.map_ij2xy([l[0][3], l[0][2]])]) for l in linesP]
        hls = [hl for hl in hls if abs(np.dot(hl.v, np.array([1, 0]))/np.dot(hl.v,
        ↪  hl.v)) < 0.1 or abs(np.dot(hl.v, np.array([0,1]))/np.dot(hl.v, hl.v)) <
        ↪  0.1]
        hls = [hl for hl in hls if hl.len() > 0.4]

        hls_clustered = create_new_pls_dbscan(hls, self.wp.params)
        self.wp.proc_new(hls_clustered, self.start_point)

        for i in range(0, len(hls)):
            l = hls[i]
            cv2.line(debug_map, tuple(self.map_xy2ij(l.p0).astype(np.int)[::-1]),
            ↪  tuple(self.map_xy2ij(l.p1).astype(np.int)[::-1]), (20,150,20), 2,
            ↪  cv2.LINE_AA)
            ress.append(create_rviz_marker_from_line(l, i, "hls", 0.5, 0, 0.5, 0.01))

        for i in range(0, len(hls_clustered)):
            ress.append(create_rviz_marker_from_line(hls_clustered[i], i,
            ↪  "hls_clustered", 0, 0, 1, 0.02))

        for i in range(0, len(self.wp.state_obj.walls)):
            ress.append(create_rviz_marker_from_line(self.wp.state_obj.walls[i].line,
            ↪  i, "walls", *((1, 0) if self.wp.state_obj.walls[i].state ==
            ↪  WallStates.UPDATABLE else (0, 1)), 0, 0.05))

        print("self.wp.state_ob.walls", self.wp.state_obj.walls)

    self.markers_arr_pub.publish(MarkerArray(markers=ress))

    debug_map[:, :, 0] = self.map_a.copy()
    # self.map_a = dilate
    self.map_img_pub.publish(self.cv_bridge.cv2_to_imgmsg(\
np.flip(debug_map, 0), "bgr8")) # mono
    return None
```

С использованием дальномера можно упростить данный процесс или дополнить его.

Следующая подзадача — поиск пожаров и пострадавших. Здесь все стандартно:

- Применение масок для нахождения пожаров и пострадавших.
- Фильтрация объектов по площади.

- Определение центров объектов в кадре.
- Преобразование координат объекта относительно карты ArUco.

```
def on_frame(self, frame, mask_floor, hsv: Optional[np.ndarray] = None):
    if hsv is None:
        hsv = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)

    # Создаем маски для нахождения пожаров и пострадавших
    debug = frame.copy()
    mask_overlay = self.mask_overlay(hsv)

    mask_blue = self.blue_overlay(hsv)

    # Создаем маску для пола площадки
    contours_floor = cv2.findContours(
        mask_floor, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE
    )[-2]

    cnt_floor = sorted(contours_floor, key=cv2.contourArea)[-1]

    mannualy_contour = []

    convex_floor = cv2.convexHull(cnt_floor, returnPoints=False)
    defects = cv2.convexityDefects(cnt_floor, convex_floor)

    if defects is not None:
        for i in range(defects.shape[0]):
            s, e, f, d = defects[i, 0]
            start = tuple(cnt_floor[s][0])
            end = tuple(cnt_floor[e][0])
            far = tuple(cnt_floor[f][0])

            dst = self.distance(start, end)

            mannualy_contour.append(start)
            if dst >= 40:
                mannualy_contour.append(far)
            mannualy_contour.append(end)

    mannualy_contour = np.array(mannualy_contour).reshape((-1, 1, 2)).astype(np.int32)
    cv2.drawContours(debug, [mannualy_contour], 0, (0, 255, 0), 3)

    mask_floor = np.zeros(mask_floor.shape, dtype="uint8")
    mask_floor = cv2.fillPoly(mask_floor, pts=[mannualy_contour], color=(255, 255,
    ↪  255))

    mask = cv2.bitwise_and(mask_overlay, mask_overlay, mask=mask_floor)
    mask_blue = cv2.bitwise_and(mask_blue, mask_blue, mask=mask_floor)

    masks = [mask, mask_blue]

    # Проходимся по маскам для нахождения пожаров и пострадавших
    for idx, m in enumerate(masks):
        contours = cv2.findContours(m, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)[-2]

        frame_vol = np.prod(frame.shape[0:2])

        # Фильтруем объекты по площади
        assert frame_vol != 0
        contours = list(
```

```
        filter(
            lambda c: (cv2.contourArea(c) / frame_vol) >= self.fire_fraction
            and (cv2.contourArea(c) / frame_vol) < 0.2,
            contours,
        )
    )

    # Находим центры объектов в кадре
    pnt_img = []
    for cnt in contours:
        M = cv2.moments(cnt)

        if M["m00"] == 0:
            continue

        pnt_img.append([int(M["m10"] / (M["m00"])), int(M["m01"] / (M["m00"]))])

        cv2.circle(debug, tuple(pnt_img[-1]), 6, (255, 0, 0), 2)

        color = (0, 255, 0) if idx == 0 else (0, 0, 255)
        cv2.drawContours(debug, [cnt], 0, color, 3)

    # Находим координаты объекта, относительно aruco_map
    if len(pnt_img) > 0:
        pnt_img = np.array(pnt_img).astype(np.float64)
        pnt_img_undist = (
            cv2.undistortPoints(
                pnt_img.reshape(-1, 1, 2), self.cm, self.dc, None, None
            )
            .reshape(-1, 2)
            .T
        )
        ray_v = np.ones((3, pnt_img_undist.shape[1]))
        ray_v[:2, :] = pnt_img_undist
        ray_v /= np.linalg.norm(ray_v, axis=0)

        if self.tf_buffer is not None:
            try:
                transform = self.tf_buffer.lookup_transform(
                    "aruco_map", "main_camera_optical", rospy.Time()
                )
            except tf2_ros.ConnectivityException:
                print("LookupException")
                return None
            except tf2_ros.LookupException:
                print("LookupException")
                return None

            t_wb = np.array(
                [
                    transform.transform.translation.x,
                    transform.transform.translation.y,
                    transform.transform.translation.z,
                ]
            )

            ray_v = np.array(
                [
                    unpack_vec(
                        tf2_geometry_msgs.do_transform_vector3(
```

```
                            Vector3Stamped(vector=Vector3(v[0], v[1], v[2])),
                            transform,
                        )
                    )
                    for v in ray_v.T
                ]
            )
            ray_o = t_wb

            pnts = [intersect_ray_plane(v, ray_o) for v in ray_v]
            [self.insert_fire(p[:2], idx) for p in pnts if p is not None]
```

Сброс капсул для тушения пожара осуществляется с помощью библиотеки `pigpiod` и следующего класса.

```
if USE_GPIO:
    import pigpio

    class GPIOHandler:
        TYPE_A_PIN = 22
        TYPE_B_PIN = 27

        def __init__(self):
            self.pi = pigpio.pi()
            self.pi.set_mode(self.TYPE_A_PIN, pigpio.OUTPUT)
            self.pi.set_mode(self.TYPE_B_PIN, pigpio.OUTPUT)

            self.clear()

        def clear(self):
            self.pi.write(self.TYPE_A_PIN, 0)
            self.pi.write(self.TYPE_B_PIN, 0)

        def push_object(self, type_o):
            a_pin = (1 if type_o == 'A' else 0)
            b_pin = (not a_pin)

            self.pi.write(self.TYPE_A_PIN, a_pin)
            self.pi.write(self.TYPE_B_PIN, b_pin)
            time.sleep(0.005)
            self.clear()
    gpio_obj = GPIOHandler()
```

И наконец, движение по траектории для обнаружения возгораний внутри помещения проще всего выполнять по уже известной траектории, которая может немного изменяться исходя из формы стены. Однако у участников есть возможность менять код перед попыткой, так что было достаточно использовать массив с критическими точками.

```
        clover_path = [0, 1, 0, 4, 7, 4, 7, 0, 1, 0]
        if not FLY_BY_SQUARE:
            clover_path = [0, 2, 0, 3, 1, 3.5, 1.5, 3, 2, 4, 4, 4, 4, 0.9, 4, 2.5,
            ↪  6.5, 2.5, 6.7, 0.5, 6.5, 3.5, 4.5, 3, 4.5, 4, 2, 4, 1.5, 3, 0, 3, 0,
            ↪  2]


        # Пролетаем по координатам
```

```
navigate_wait(z=FLIGHT_HEIGHT, speed = 0.6, frame_id='body', auto_arm=True)
rospy.sleep(2)
print('takeoff')

t = get_telemetry(frame_id='aruco_map')
land_position = (t.x, t.y)
rospy.sleep(1)

for i in range(0, len(clover_path), 2):
    if i == 4:
        # При подлете к рабочей зоне, включаем распознавание пожаров, стен и
        ↪  пострадавших
        handler.enable()

    navigate_wait(x=clover_path[i], y=clover_path[i+1], z=FLIGHT_HEIGHT,
    ↪  speed=0.3, frame_id="aruco_map")
    print("go to next point...")

# Выключаем распознавание
handler.disable()

navigate_wait(x=land_position[0], y=land_position[1], z=FLIGHT_HEIGHT,
↪  speed=0.3, frame_id="aruco_map")
rospy.sleep(3)
land()

# Отчет
handler.fires.report()
handler.wd.report()
```

Инженерная часть

Перед началом разработки устройства необходимо ознакомиться с техническим заданием и обратить внимание на имеющиеся ограничения:

- Временные рамки на проектирование, изготовление и доработку устройства.
- Электронную компонентную базу (Arduino, сервопривод в количестве 2 шт.).
- Вес (200 г).
- Технологии производства:
- Для минимизации использования поддержек необходимо выбрать правильное направление и ориентацию деталей, а также можно использовать геометрические формы, требующие минимальное число поддержек.
- Для избежания деформации и провисания пластика во время печати необходимо обеспечить достаточную толщину стенок деталей и их правильное расположение.
- Для гарантированного соединения деталей между собой необходимо расположить детали на печатном столе так, чтобы поверхности и отверстия, с помощью которых детали соединяются, имели максимальную точность.
- Для минимизации времени печати и использования материала необходимо подобрать оптимальное расположение деталей на печатной платформе (закладывать минимальное расстояние между деталями; сторону детали, имеющую максимальный линейный размер, располагать горизонтально на поверхности стола и т. п.).
- Перед началом работы с принтером необходимо произвести калибровку печат-

ной платформы для обеспечения правильного прижима первого слоя путем последовательной проверки зазора между соплом и столом по нескольким противолежащим точкам в углах стола.

- Перед запуском основной печати для настройки температуры необходимо произвести печать тестовой модели из необходимого пластика.
- Использовать подходящие настройки скорости печати для конкретных деталей. Чтобы определить оптимальную скорость перемещений на холостом ходе для имеющегося принтера, необходимо распечатать тестовую модель при различных скоростях движения, начиная со 100 мм/с и изменяя ее с приростом 5 мм/с. Скорость печати можно увеличивать, если качество поверхности приемлемое, и уменьшать, если качество 3D печати ухудшается. Необходимо обращать внимание на такие дефекты, как несовпадение слоев.

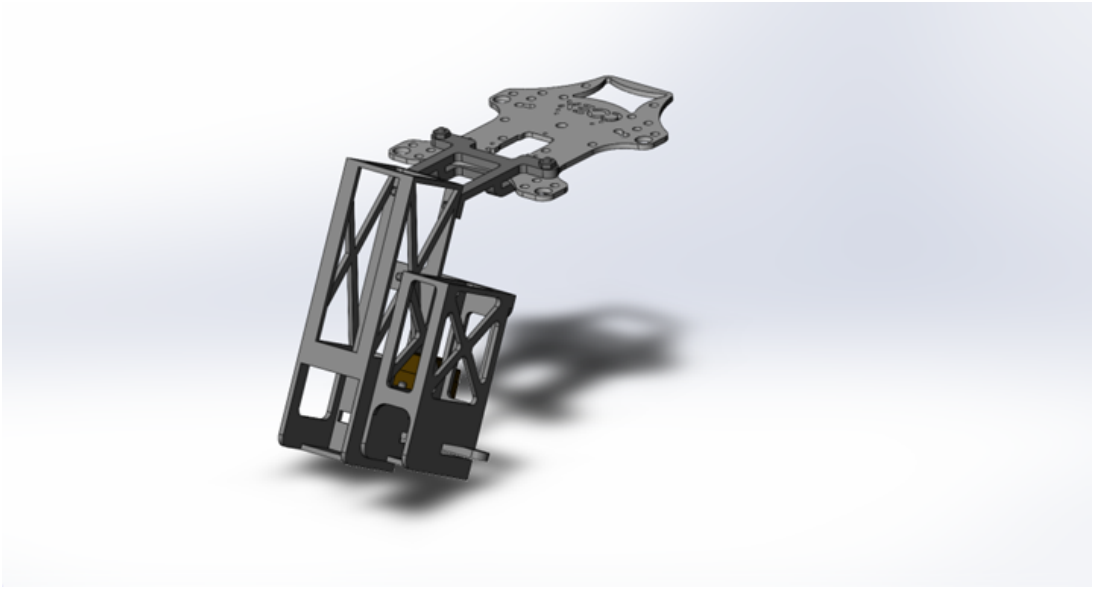

Рис. VI.2.3. Вид в изометрии

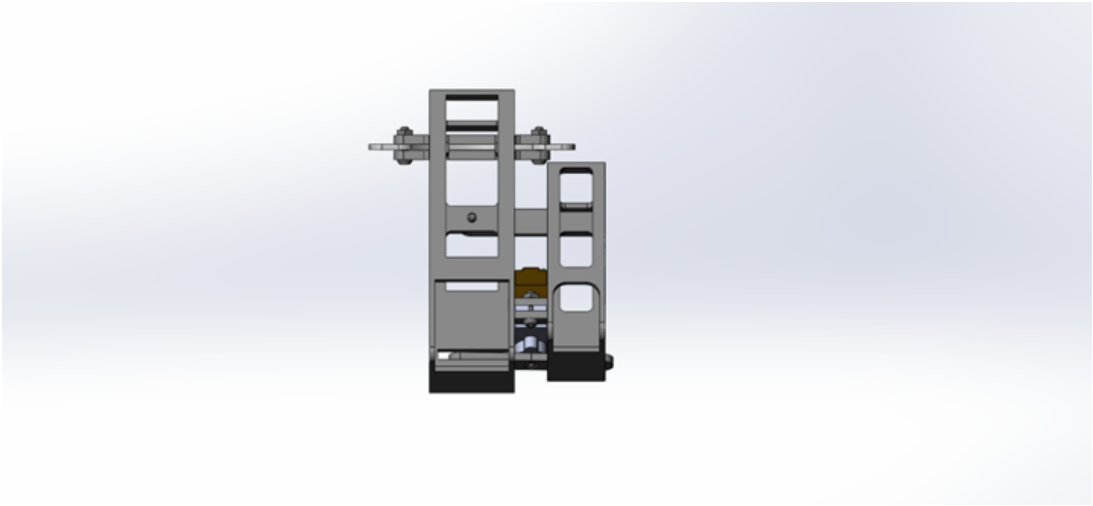

Рис. VI.2.4. Вид спереди

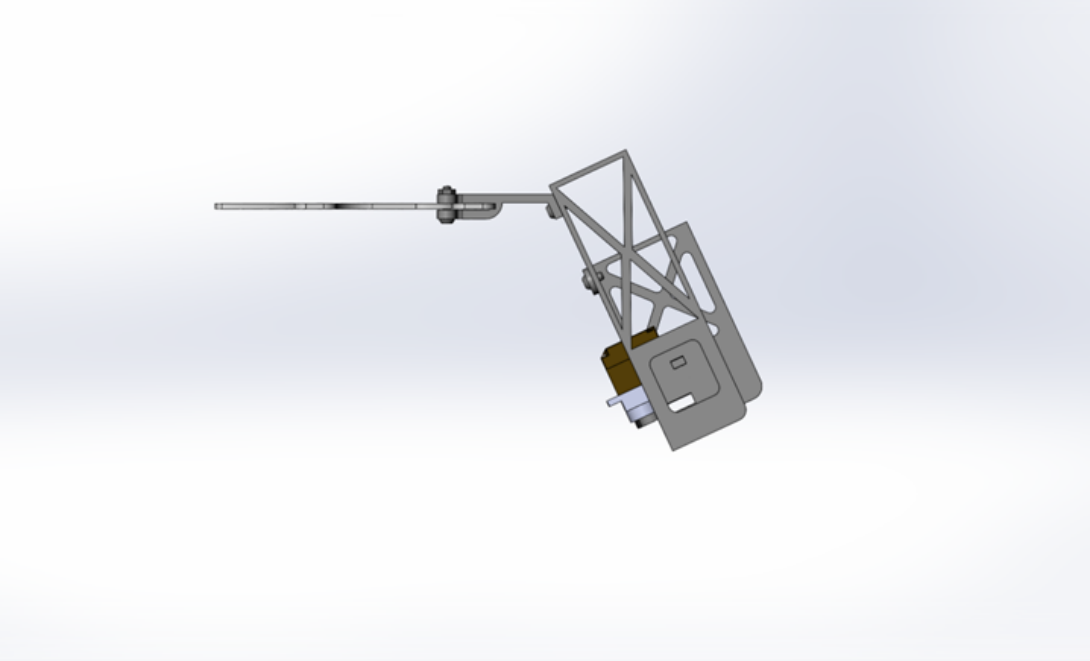

Рис. VI.2.5. Вид сбоку

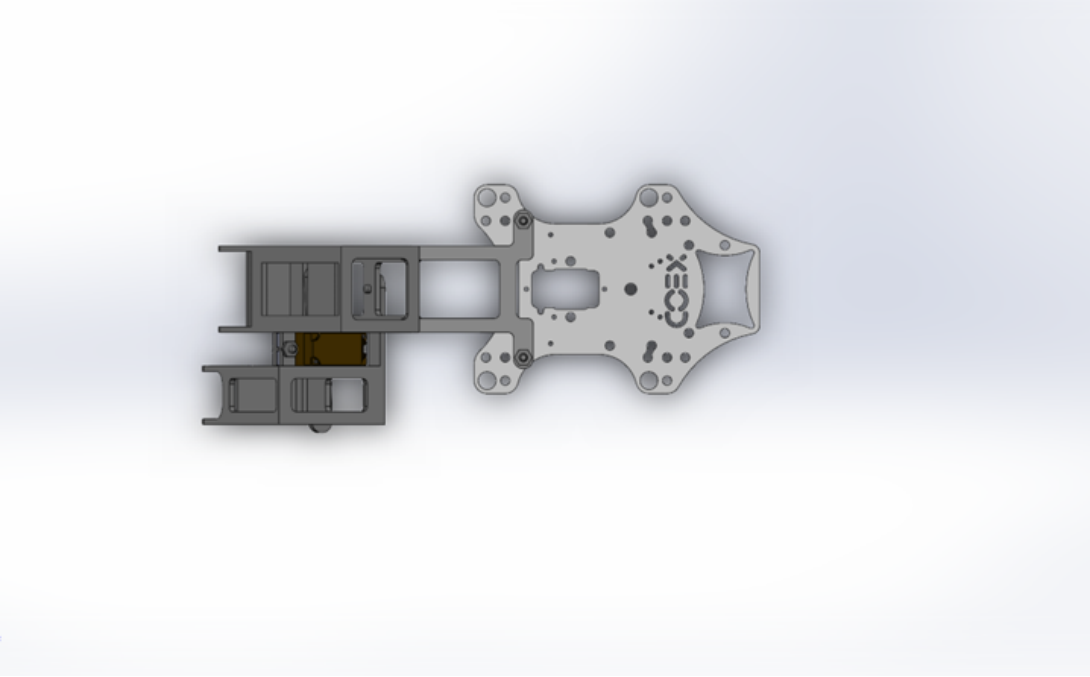

Рис. VI.2.6. Вид сверху

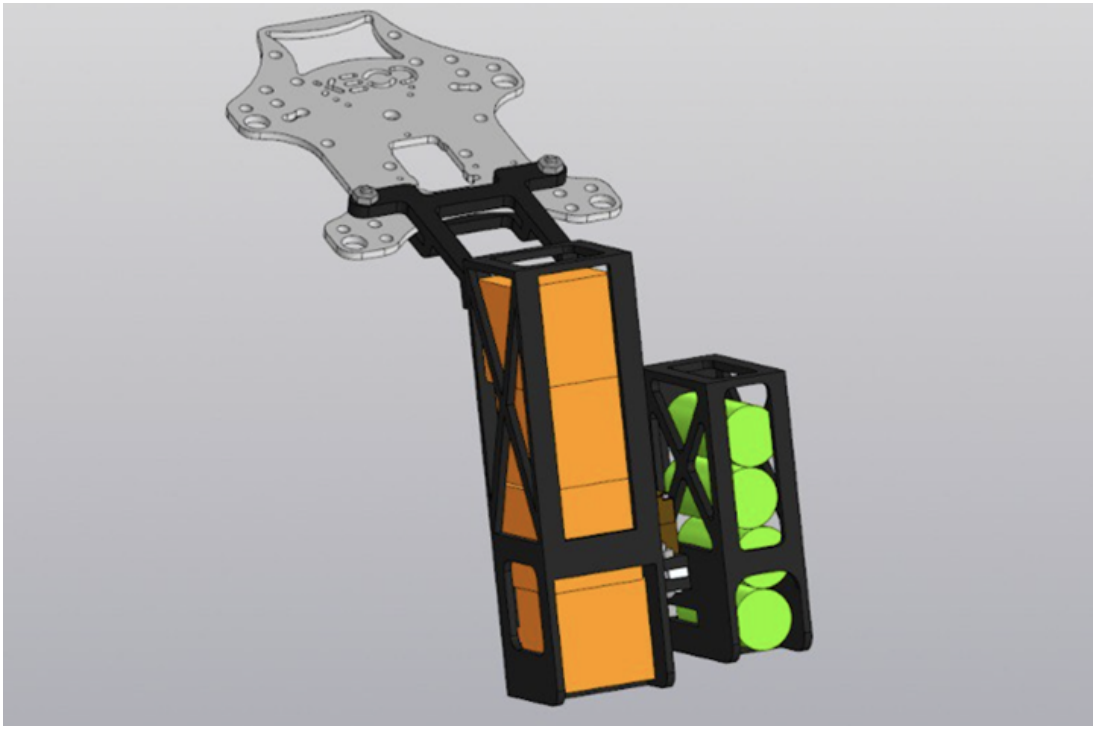

Рис. VI.2.7. Функциональное устройство с целевой нагрузкой (капсулы)

Программный код для эксплуатации устройства (Arduino)

```
#include <Servo.h>

const int P1 = 35;
const int P0 = 90;
const int P2 = 155;

boolean b1 = false;
boolean b2 = false;

boolean b1_prev = false;
boolean b2_prev = false;

Servo servo;

void setup() {
    servo.attach(9);
    servo.write(P0);
    pinMode(11, INPUT);
    pinMode(12, INPUT);
}

void loop() {
    b1 = digitalRead(11);
    b2 = digitalRead(12);

    if (b1 == 1 and b1_prev == 0)
    {
        servo.write(P1);
        delay(1000);
        servo.write(P0);
    } else if (b2 == 1 and b2_prev == 0)
```

```
32      {
33          servo.write(P2);
34          delay(1000);
35          servo.write(P0);
36      }
37      b1_prev = digitalRead(11);
38      b2_prev = digitalRead(12);
39  }
```

Материалы для подготовки

- Статьи, видеолекции по тематике профиля на сайте документации платформы «Клевер»: https://clover.coex.tech/.
- Видеокурс по сборке, настройке и программированию квадрокоптера: https://www.youtube.com/watch?v=GJvucA2igME&list=PLpWUdRMCiBWaSy0ZvTUttDJkVnUF3p2TD.
- Среда симуляции платформы Клевер: https://clover.coex.tech/ru/simulation.html.
- Работа с OpenCV при помощи Python: https://github.com/sfalexrog/coex_kb/blob/master/kb014_opencv_python.md.
- Программирование на Python: https://stepik.org/course/67/promo.
- Введение в ROS: https://stepik.org/course/3222/promo.
- Введение в Git: https://ru.hexlet.io/courses/intro_to_git.
- Шоу дронов: от создания анимации до запуска роя: https://stepik.org/course/123912/syllabus.