П. В. БАЛАБАНОВ, А. Г. ДИВИН, Д. А. ЛЮБИМОВА

ПРОГРАММИРОВАНИЕ БЕСПИЛОТНОГО ЛЕТАТЕЛЬНОГО АППАРАТА МУЛЬТИРОТОРНОГО ТИПА

python

Тамбов
Издательский центр ФГБОУ ВО «ТГТУ»
2023

Министерство науки и высшего образования Российской Федерации

**Федеральное государственное бюджетное
образовательное учреждение высшего образования
«Тамбовский государственный технический университет»**

П. В. БАЛАБАНОВ, А. Г. ДИВИН, Д. А. ЛЮБИМОВА

ПРОГРАММИРОВАНИЕ БЕСПИЛОТНОГО ЛЕТАТЕЛЬНОГО АППАРАТА МУЛЬТИРОТОРНОГО ТИПА

Утверждено Ученым советом университета в качестве учебного пособия
для студентов, обучающихся по программам бакалавриата и магистратуры
15.03.06, 15.04.06 «Мехатроника и робототехника»

Учебное электронное издание

Тамбов
Издательский центр ФГБОУ ВО «ТГТУ»
2023

УДК 519.816
ББК 22.17
Б20

Рецензенты:

Доктор технических наук, заведующий лабораторией медицинских VR тренажерных систем для обучения, диагностики и реабилитации ФГБОУ ВО «ТГТУ»
А. Д. Обухов

Главный конструктор «Тамбовский завод «Электроприбор»
В. А. Никитин

Балабанов, П. В.

Б20 Программирование беспилотного летательного аппарата мультироторного типа [Электронный ресурс] : учебное пособие / П. В. Балабанов, А. Г. Дивин, Д. А. Любимова. – Тамбов : Издательский центр ФГБОУ ВО «ТГТУ», 2023. – 1 электрон. опт. диск (CD-ROM). – Системные требования : ПК не ниже класса Pentium II ; CD-ROM-дисковод ; 2,0 Mb ; RAM ; Windows 95/98/XP ; мышь. – Загл. с экрана.

ISBN 978-5-8265-2689-7

Содержит методические указания к практическим и лабораторным работам дисциплины «Программирование мехатронных систем». Рассмотрены следующие вопросы: подготовка технического задания на разработку программного обеспечения мехатронных систем; разработка программного обеспечения для управления мехатронных систем на примере квадрокоптера; получение и обработка изображений с камер видимого и инфракрасного диапазона спектра излучения; тестирование и расчет надежности программного обеспечения.

Предназначено для студентов, обучающихся по программам бакалавриата и магистратуры 15.03.06, 15.04.06 «Мехатроника и робототехника». Может быть интересно начинающим программистам-робототехникам, студентам, а также всем, кто интересуется вопросами программирования дронов с использованием языка python.

УДК 519.816
ББК 22.17

ISBN 978-5-8265-2689-7

ВВЕДЕНИЕ

В современном мире информационные технологии развиваются высокими темпами, поэтому написание полезных и актуальных учебных пособий по программированию, в частности на популярном языке python, является крайне сложной задачей. Процесс написания, апробации, редактирования и последующего издания может растянуться на месяцы, в то время как описываемые технологии получат обновление, а сами материалы пособия уже устареют и станут не актуальными. Поэтому в данной работе мы постарались сделать акцент не на самом языке программирования, а на его применении при решении конкретных узкоспециализированных задач в области управления беспилотными летательными аппаратами. Стоит сразу оговориться, что приведенные примеры применимы к квадракоптерам Tello и вряд ли получится применить рассмотренные инструменты и методы языка программирования к широкому спектру беспилотных летательных аппаратов (БПЛА). Однако изложенный материал может быть полезен специалистам начального уровня, постигающим достаточно трудную науку управления мехатронными и робототехническими системами, частным случаем которых являются летательные аппараты мультироторного типа. Пособие включает ряд практических работ, в процессе выполнения которых обучающиеся, помимо изложенных инструментов программирования, должны будут изучить принципы работы систем технического зрения, применяемых в составе мехатронных и робототехнических систем, работающих в видимом и ИК-диапазонах спектра излучения, других датчиков и сенсоров. Ознакомление со стандартами ЕСПД и их применение при разработке программной документации позволит получить востребованные навыки технического писательства. Для успешного освоения изложенных материалов мы рекомендуем ознакомиться с дополнительной литературой по методам и средствам программирования на python, системам технического зрения, методам теории автоматического управления, список которой приведен в конце пособия.

ТЕХНИЧЕСКИЕ КОМПОНЕНТЫ БПЛА. КРАТКИЕ СВЕДЕНИЯ

Беспилотный летательный аппарат (БПЛА) – это искусственный мобильный объект (летательный аппарат), как правило, многоразового использования, не имеющий на борту экипажа (человека-пилота) и способный самостоятельно целенаправленно перемещаться в воздухе для выполнения различных функций в автономном режиме (с помощью собственной управляющей программы) или посредством дистанционного управления (осуществляемого человеком-оператором со стационарного или мобильного пульта управления).

Несущей конструкцией БПЛА является планер. Планеры отличаются формой крыла, фюзеляжа, условиями применения. Неотъемлемой частью БПЛА является бортовая система электропитания, в состав которой входят источники тока, аппаратура управления, сеть питания потребителей. В качестве источников электроэнергии часто используют бортовые электрогенераторы и аккумуляторные батареи. В качестве силовых установок БПЛА применяют различного типа двигатели, прочие устройства и агрегаты, обеспечивающие силу тяги, подъемную силу для полета и ускорения аппарата. Важнейшей частью БПЛА является система управления, представляющая комплекс технических, программно-алгоритмических средств, обеспечивающих изменение параметров движения летательного аппарата в требуемом направлении. В основе процесса управления лежит информация о задачах управления (заданной цели) и текущем состоянии системы, получаемая от различных датчиков, входящих в состав системы навигации.

Исторически раньше всего появились инерциальные системы навигации. Их достоинством является непрерывная динамичная выдача пользователю полного навигационного решения (координаты, скорость, ускорения, угловая ориентация), возможность выдачи информации с высокой частотой, независимость от внешних источников информации. Однако инерциальные системы навигации обладают недостатком – ошибка в определении навигационных параметров накапливается с течением времени, а точность выходной информации зависит от точности чувствительных элементов. В качестве чувствительных элементов

применяются прецизионные дорогостоящие гироскопы и акселерометры, которые сильно усложняют и удорожают систему.

Точность выходной навигационной информации напрямую зависит от характеристик чувствительных элементов, входящих в состав системы, – акселерометров, гироскопов, магнетометров, высотомеров и пр. Акселерометр – прибор, измеряющий проекцию кажущегося ускорения (разности между истинным ускорением объекта и гравитационным ускорением). Акселерометры бывают механическими, электронными, пьезоэлектрическими, термальными.

Гироскоп – устройство, способное реагировать на изменение углов ориентации тела, на котором оно установлено, относительно инерциальной системы отсчета. Конструкцию гироскопа принято разделять на две группы по принципу действия: механические, оптические.

Датчики, работающие по принципу гироскопа, являются неотъемлемым оборудованием в авиации. Два гироскопа устанавливаются на крыльях самолета, благодаря чему можно получать информацию о его повороте вокруг вертикальной оси. Распространенные сейчас БПЛА имеют три гироскопа, без которых управление летательным аппаратом и его точное балансирование было бы невозможным.

Движение БПЛА осуществляется путем контроля трех режимов углового движения – тангажа, рыскания, крена. Трехконтурный PID-регулятор отслеживает положение БПЛА, вырабатывая курсовой угол. Автопилот БПЛА осуществляет выработку управляющих команд в виде ШИМ (широтно-импульсно-модулированных) сигналов, сообразно законам управления, заложенным в его вычислитель.

Сложность управления беспилотным летательным аппаратом – один из сдерживающих факторов их распространения. Современные модели, оборудованные интегрированными навигационными системами, доступны по приемлемым ценам. Их эксплуатация является коммерчески выгодной ввиду повышения точности и оперативности выполняемых ими работ, проводимых с учетом полученных сведений. При этом, в зависимости от целей применения, в качестве целевой нагрузки БПЛА могут быть различного типа камеры (видимого, инфракрасного диапазона, гиперспектральная), локационное оборудова-

ние, различные датчики и сенсоры. Беспилотные летательные аппараты можно использовать как платформы для установки специализированного навесного оборудования, например систем опрыскивания сельскохозяйственных растений.

При проектировании БПЛА отдельный интерес представляет разработка системы управления, которую условно можно разделить на наземную (стационарную или мобильную) и бортовую. Наземные (стационарные) системы обычно располагаются в специально оборудованных помещениях. Мобильные системы контроля и навигации вместе с оператором могут находиться в фургоне, мобильном передвижном пункте, либо же в ручном кейсе, которые быстро и легко можно развернуть в полноценные станции управления.

В состав наземных (стационарных или мобильных) систем обычно входят следующие компоненты:

– система приема сигналов от БПЛА;

– система отправки сигналов воздушному судну;

– системы приема и обработки изображения на мониторе системы приема тепловизионного, инфракрасного изображения;

– пульт управления внешнего пилота (включая системы программирования полетов и полетного задания).

В комплекс бортовых подсистем управления на стороне БПЛА входят:

– устройства получения видовой информации;

– системы для связи через спутники (Глонас/GPS);

– устройства телеметрии и передачи на пункт управления информации о параметрах положения судна в пространстве, скорости полета, длительности полета и т.д.;

– устройства командно-навигационной радиолинии с антенно-фидерным устройством;

– бортовая вычислительная машина;

– подсистемы для хранения полетной информации, а также фото- и видеоизображений.

Автоматическая бортовая система управления БПЛА должна быть способной решать основные следующие задачи:

– стабилизация параметров движения объекта применительно к внешним помехам различной природы;

– анализ внешних данных бортовыми средствами и определение приоритетной цели в зависимости от поставленной перед БПЛА задачи;

– расчет оптимальной траектории движения в целях уменьшения времени движения и расхода ресурсов БПЛА;

– контроль правильности удержания траектории;

– обеспечение отказоустойчивости объекта управления или компенсация изменений его характеристик бортовыми средствами;

– выполнение вычислительных операций большого объема в реальном масштабе времени для реализации алгоритмов управления БПЛА.

Указанные задачи решаются с использованием программно-аппаратных средств – полетных контроллеров.

Таким образом, проектирование БПЛА является сложной, комплексной задачей, требующей компетенций в различных областях знания – систем управления, датчиков и сенсоров, обработки больших данных и пр. В рамках данного учебного пособия мы рассмотрим некоторые задачи, связанные с разработкой системы управления БПЛА квадракоптерного типа, на примере дрона tello. Разработанные нами алгоритмы управления будут реализованы на наземной рабочей станции – персональном компьютере, управляющие команды будут передаваться по беспроводному каналу связи на полетный контроллер квадракоптера. Такой подход обусловлен достаточно миниатюрными размерами дрона и ограниченными вычислительными ресурсами его полетного контроллера, что не позволяет нам использовать программно-алгоритмические средства обработки изображений с камеры дрона непосредственно в самом его полетном контроллере. Помимо задач собственно управления дроном, мы рассмотрим примеры идентификации объектов контроля с помощью камер, получающих изображение в видимом и ИК-спектрах излучения, что представляет интерес при практическом применении дронов.

Практическая работа 1

ТЕХНИЧЕСКОЕ ЗАДАНИЕ НА РАЗРАБОТКУ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ ДЛЯ УПРАВЛЕНИЯ МЕХАТРОННОЙ СИСТЕМОЙ: ТРЕБОВАНИЯ ДЕЙСТВУЮЩИХ СТАНДАРТОВ ЕСПД, СОСТАВ, РАЗРАБОТКА

Цель работы – ознакомиться с требованиями стандартов ЕСПД на разработку технического задания, научиться применять ГОСТ 19.106–78 при разработке технического задания.

Методические указания

В ходе выполнения практической работы необходимо решить следующие задачи:

– изучить требования ГОСТ 19.106–78 к оформлению и содержанию технического задания;

– обсудить содержание основных разделов технического задания;

– обсудить основные требования к разрабатываемой системе управления мехатронным устройством;

– сформировать перечень функциональных характеристик разрабатываемого программного средства.

Основные требования к оформлению и содержанию технического задания изложены в ГОСТ 19.106–78.

Техническое задание оформляют в соответствии с ГОСТ 19.106–78 на листах формата А4 и А3. Номера листов (страниц) проставляются в верхней части листа над текстом. Лист утверждения и титульный лист оформляют в соответствии с ГОСТ 19.104–78.

Техническое задание должно содержать следующие разделы:

- введение;

- основания для разработки;
- назначение разработки;
- требования к программе или программному изделию;
- требования к программной документации;
- технико-экономические показатели;
- стадии и этапы разработки;
- порядок контроля и приемки;
- в техническое задание допускается включать приложения.

В зависимости от особенностей программы или программного изделия допускается уточнять содержание разделов, вводить новые разделы или объединять отдельные из них.

В разделе «Введение» указывают наименование, краткую характеристику области применения программы или программного изделия и объекта, в котором используют программу или программное изделие.

В разделе «Основания для разработки» должны быть указаны:

- документ (документы), на основании которых ведется разработка;
- организация, утвердившая этот документ, и дата его утверждения;
- наименование и(или) условное обозначение темы разработки.

В разделе «Назначение разработки» должно быть указано функциональное и эксплуатационное назначение программы или программного изделия.

Раздел «Требования к программе или программному изделию» должен содержать следующие подразделы:

- требования к функциональным характеристикам;
- требования к надежности;
- условия эксплуатации;
- требования к составу и параметрам технических средств;
- требования к информационной и программной совместимости;
- требования к маркировке и упаковке;
- требования к транспортированию и хранению;
- специальные требования.

В подразделе «Требования к функциональным характеристикам» должны быть указаны требования к составу выполняемых функций, организации входных и выходных данных, временным характеристикам.

В подразделе «Требования к надежности» должны быть указаны требования к обеспечению надежного функционирования (обеспечения устойчивого функционирования, контроль входной и выходной информации, время восстановления после отказа и т.п.).

В подразделе «Условия эксплуатации» должны быть указаны условия эксплуатации (температура окружающего воздуха, относительная влажность и т.п. для выбранных типов носителей данных), при которых должны обеспечиваться заданные характеристики, а также вид обслуживания, необходимое количество и квалификация персонала.

В подразделе «Требования к составу и параметрам технических средств» указывают необходимый состав технических средств с указанием их основных технических характеристик.

В подразделе «Требования к информационной и программной совместимости» должны быть указаны требования к информационным структурам на входе и выходе и методам решения, исходным кодам, языкам программирования и программным средствам, используемым программой.

При необходимости должна обеспечиваться защита информации и программ.

В подразделе «Требования к маркировке и упаковке» в общем случае указывают требования к маркировке программного изделия, варианты и способы упаковки.

В подразделе «Требования к транспортированию и хранению» должны быть указаны для программного изделия условия транспортирования (носителей), места хранения, условия хранения, условия складирования, сроки хранения в различных условиях.

В разделе «Требования к программной документации» должен быть указан предварительный состав программной документации и при необходимости специальные требования к ней.

В разделе «Технико-экономические показатели» должны быть указаны: ориентировочная экономическая эффективность, предполагаемая годовая потребность, экономические преимущества разработки по сравнению с лучшими отечественными и зарубежными образцами или аналогами.

В разделе «Стадии и этапы разработки» устанавливают необходимые стадии разработки, этапы и содержание работ (перечень программных документов, которые должны быть разработаны, согласованы и утверждены), а также, как правило, сроки разработки и определяют исполнителей.

В разделе «Порядок контроля и приемки» должны быть указаны виды испытаний и общие требования к приемке работы.

В приложениях к техническому заданию при необходимости приводят:

- перечень научно-исследовательских и других работ, обосновывающих разработку;
- схемы алгоритмов, таблицы, описания, обоснования, расчеты и другие документы, которые могут быть использованы при разработке;
- другие источники разработки.

Требования к программным документам, выполненным печатным способом, установлены в ГОСТ 19.106–78.

Программный документ выполняют одним из следующих печатных способов:

- машинописным – на одной стороне листа через два интервала; допускается через один или полтора интервала;
- типографским – в соответствии с требованиями, предъявленными к изданиям, изготавливаемым типографским способом.

Программные документы оформляют:

- на листах формата А4 (ГОСТ 2.301–68) – при изготовлении документа машинописным или рукописным способом (форма 1). Допускается оформление на листах формата А3 (форма 2);
- на листах типографических форматов – при изготовлении документа типографским способом.

Материалы программного документа располагают в следующей последовательности:

- титульная часть:
  - лист утверждения (не входит в общее число листов документа);
  - титульный лист (первый лист документа);
- информационная часть:
  - аннотация;
  - лист содержания;
- основная часть:
  - текст документа (с рисунками, таблицами и т.п.);
  - приложения;
  - перечень терминов;
  - перечень сокращений;
  - перечень рисунков;
  - перечень таблиц;
  - предметный указатель;
  - перечень ссылочных документов;
  - перечень символов и числовых коэффициентов;
- часть регистрации изменений:
  - лист регистрации изменений.

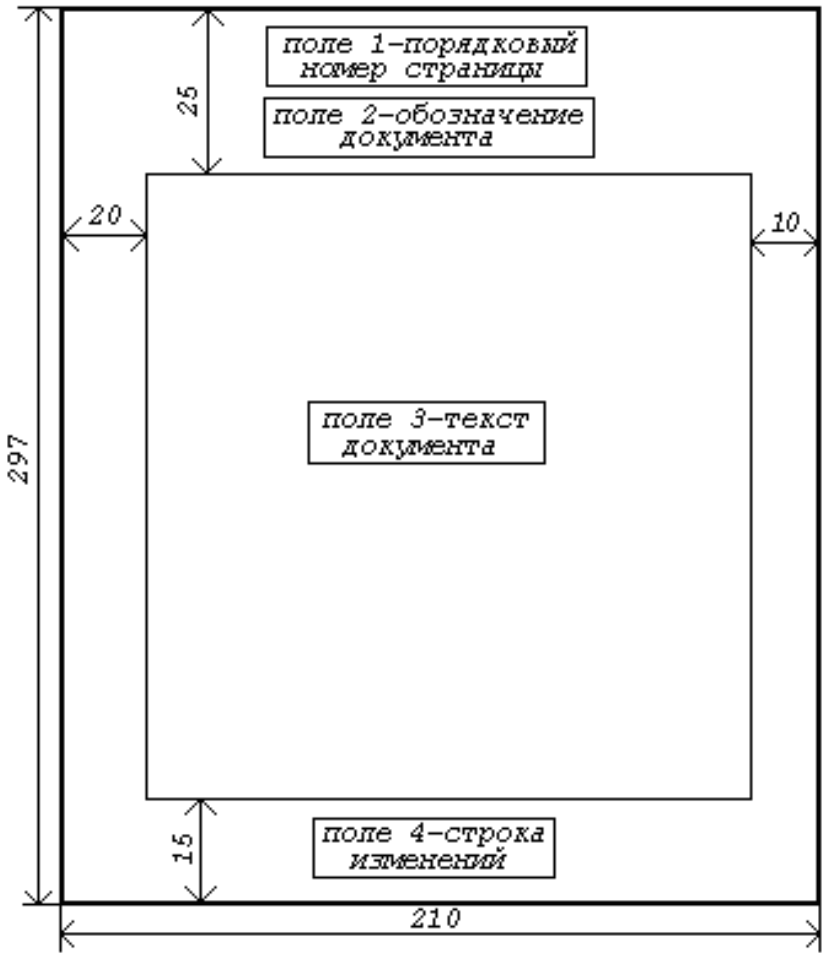
Построение документа

Лист утверждения выпускают на весь документ с обозначением первой части.

Информационная и основная части программного документа выполняются по форме 1 или 2, где:

- поле 1 – порядковый номер страницы;
- поле 2 – обозначение документа;
- поле 3 – текст документа;
- поле 4 – строка изменений; заполняется в соответствии с требованиями ГОСТ 19.604–78.

Форма 1.



Рамку (границы) формата страниц документа допускается не наносить.

Аннотацию размещают на отдельной (пронумерованной) странице с заголовком «АННОТАЦИЯ» и не нумеруют как раздел.

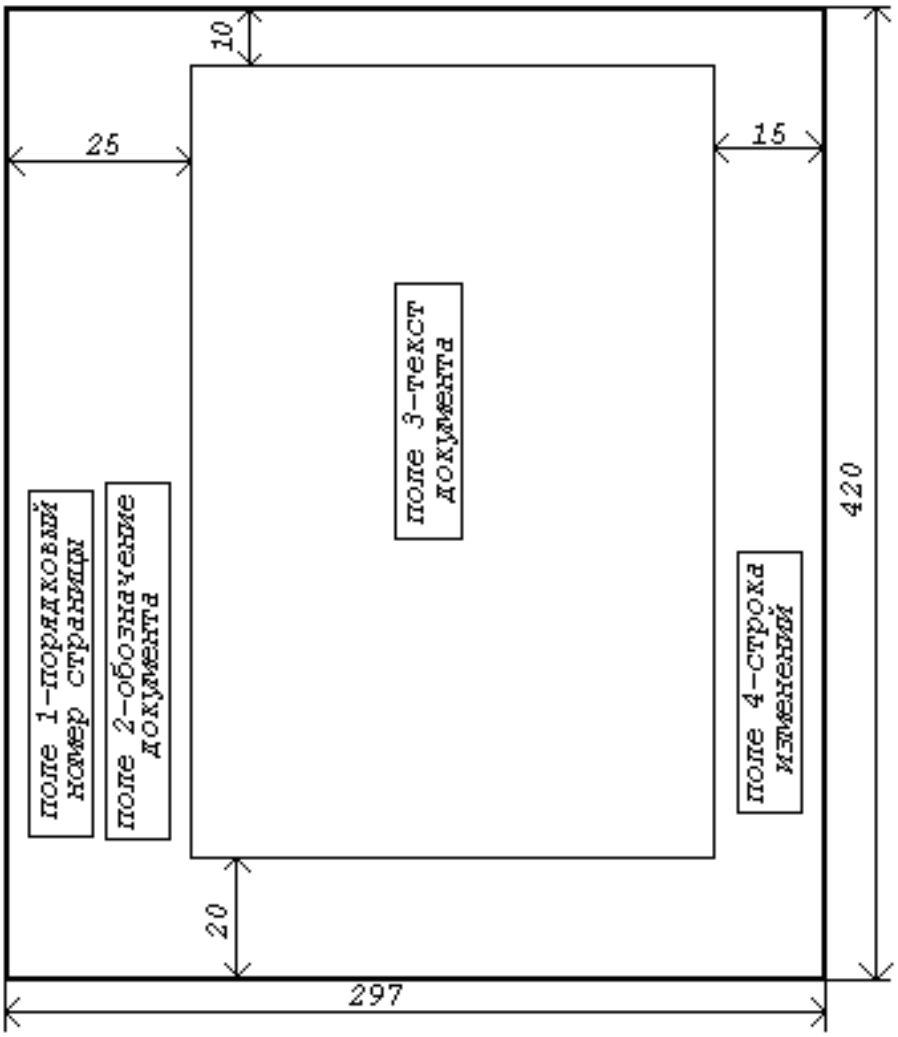
В аннотации указывают издание программы, кратко излагают назначение и содержание документа. Если документ состоит из нескольких частей, в аннотации указывают общее количество частей.

Содержание документа размещают на отдельной (пронумерованной) странице (страницах) после аннотации, снабжают заголовком «СОДЕРЖАНИЕ», не нумеруют как раздел и включают в общее количество страниц документа.

Наименования, включенные в содержание, записывают строчными буквами. Прописными должны печататься заглавные буквы и аббревиатуры.

Заголовки разделов пишут прописными буквами и размещают симметрично относительно правой и левой границ текста.

Форма 2.



Заголовки подразделов записывают с абзаца строчными буквами (кроме первой прописной).

Переносы слов в заголовках не допускаются. Точку в конце заголовка не ставят.

Если заголовок состоит из двух предложений, их разделяют точкой.

Каждый раздел рекомендуется начинать с нового листа.

Расстояние между заголовком и последующим текстом, а также между заголовками раздела и подраздела должно быть равно двум интервалам.

Разделы, подразделы, пункты и подпункты следует нумеровать арабскими цифрами с точкой.

В пределах раздела должна быть сквозная нумерация по всем подразделам, пунктам и подпунктам, входящим в данный раздел.

Нумерация подразделов включает номер раздела и порядковый номер подраздела, входящего в данный раздел, разделенные точкой (2.1; 3.1 и т.д.).

При наличии разделов и подразделов к номеру подраздела после точки добавляют порядковый номер пункта и подпункта (3.1.1, 3.1.1.1 и т.д.).

Текст документа должен быть кратким, четким, исключающим возможность неверного толкования.

Иллюстрации могут быть расположены в тексте документа и(или) в приложениях.

Иллюстрации, если их в данном документе более одной, нумеруют арабскими цифрами в пределах всего документа.

Ссылки на иллюстрации дают по типу: «рис. 12» или «(рис. 12)».

Формулы в документе, если их более одной, нумеруются арабскими цифрами, номер ставят с правой стороны страницы, в скобках на уровне формулы.

В пределах всего документа или его частей, в случае деления документа на части, формулы имеют сквозную нумерацию.

Цифровой материал для достижения лучшей наглядности и сравнимости показателей, как правило, следует оформлять в виде таблицы.

Таблица может иметь заголовок, который следует выполнять строчными буквами. Прописными должны печататься заглавные буквы и аббревиатуры.

Иллюстративный материал, таблицы или текст вспомогательного характера допускается оформлять в виде приложений.

Каждое приложение должно начинаться с новой страницы с указанием в правом верхнем углу слова «ПРИЛОЖЕНИЕ» и иметь тематический заголовок, который записывают симметрично тексту прописными буквами.

При наличии в документе более одного приложения все приложения нумеруют арабскими цифрами (без знака №), например, ПРИЛОЖЕНИЕ 1, ПРИЛОЖЕНИЕ 2 и т.д.

В состав основных надписей листа утверждения и титульного листа в программных документах входят следующие структурные данные:

- наименование министерства (ведомства);
- наименование документа;
- обозначение документа;
- сведения о носителе данных, на котором представляется подлинник;
- общее количество листов утверждения, объем документа;

- сведения о разработчике;
- виза нормоконтролера;
- отметка об учете и хранении;
- сведения об изменении.

Лист утверждения заполняют по форме, приведенной на чертеже ниже:

- поле 1 – наименование министерства или ведомства, в систему которого входит организация, разработавшая данный документ.

Заполняют по требованию заказчика.

Выше поля 1, в правом верхнем углу, при необходимости ставится специальная отметка (гриф секретности, указание «С предприятия не выносить» и т.п.);

- поле 2 – в левой части поля – должности и подписи лиц, согласовавших документ от организации заказчика при необходимости, в правой части поля – должности и подписи лиц, утвердивших документ от организации разработчика.

Справа от каждой подписи проставляют инициалы и фамилию лица, подписавшего документ, а ниже подписи – дату подписания.

Согласующие и утверждающие организации, а также конкретные подписи должностных лиц регламентируют министерства и ведомства;

- поле 3 – полное наименование программы или программного изделия (прописными буквами), наименование и вид документа.

Наименование документа может быть опущено или объединено с наименованием программы;

- поле 4 – обозначение документа и указание вида носителя данных.

Вид носителя данных указывают только в случае выполнения документа на носителе данных;

- поле 5 – общее количество листов утверждения, например, «Листов 3». Для одного листа поле 5 не заполняют;

- поле 6 – в правой части поля – должности и подписи руководителя организации, выпустившей документ, руководителя подразделения, разработавшего документ, руководителя разработки (разработчика), исполнителей разработки документа и нормоконтролера.

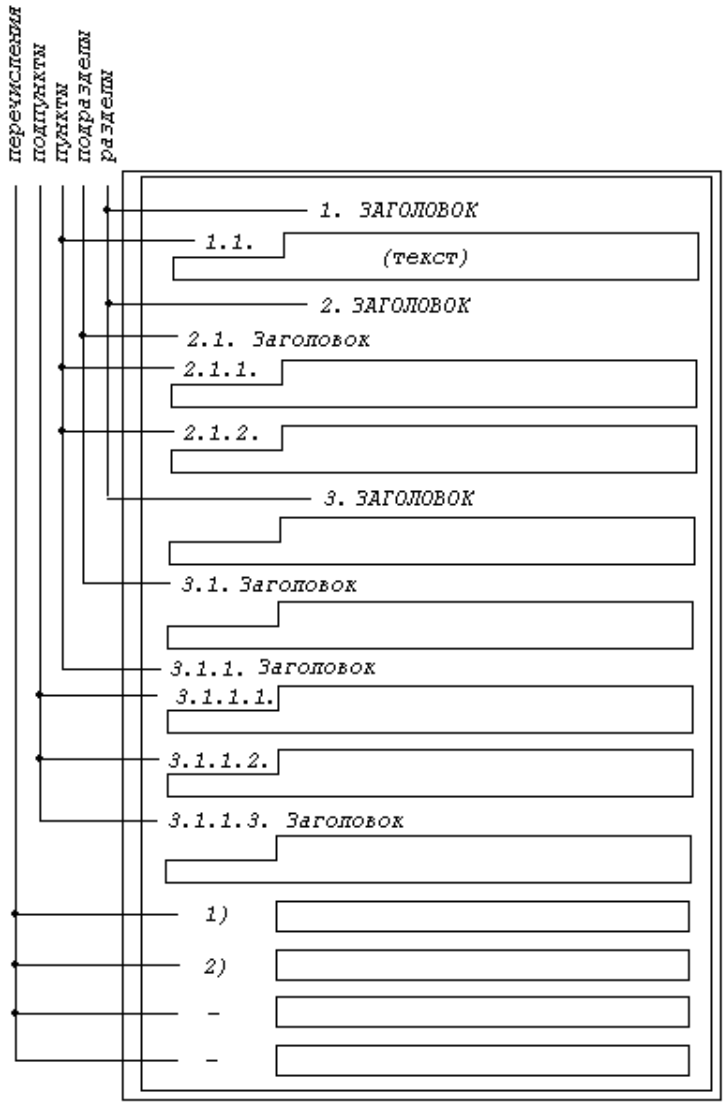
Справа от каждой подписи проставляют инициалы и фамилию лица, подписавшего документ, а ниже подписи – дату подписания.

При большом количестве согласующих подписей их размещают либо в левой части поля 2, либо в левой части поля 6.

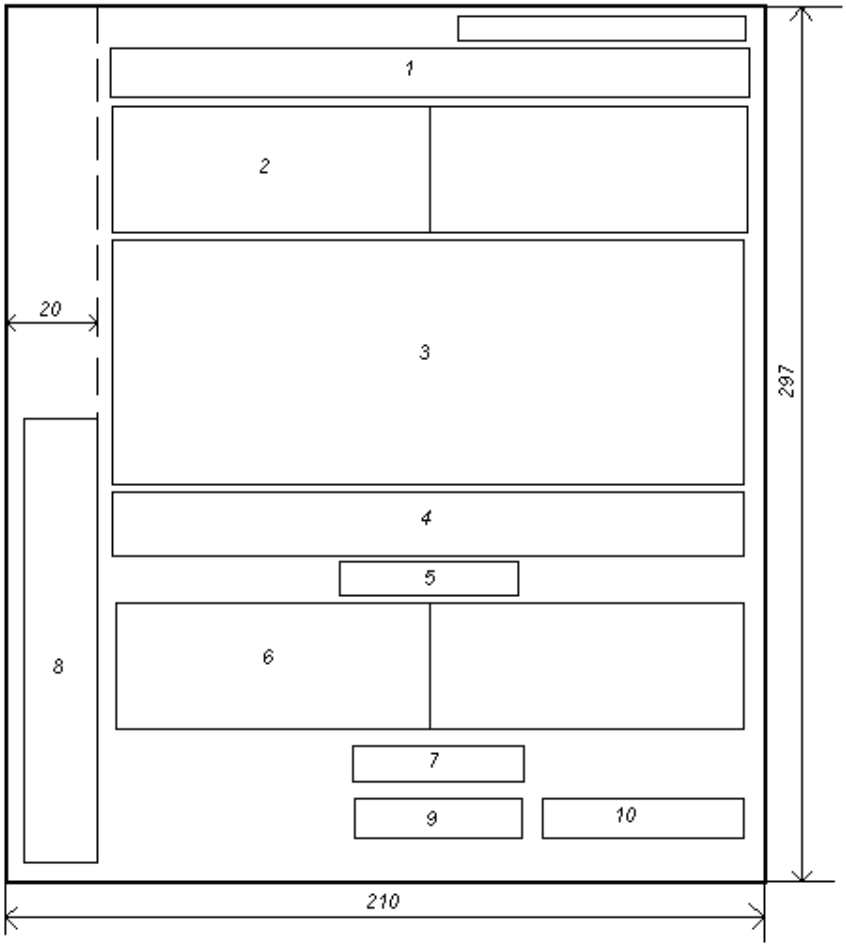
Визы других должностных лиц, если они необходимы на документе, размещают на поле для подшивки листа утверждения;

- поле 7 – год издания (утверждения) документа (без указания слова «год» или «г»);
- поле 8 – отметка об учете и хранении по ГОСТ 19.601–78;
- поле 9 – строка изменений по ГОСТ 19.604–78;
- поле 10 – литера документа.

Структура текста программного документа



Форма листа утверждения


1
2
3
4
5
6
7
8
9
10
20
297
210

Практическая работа 2

РАЗРАБОТКА ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ ДЛЯ УПРАВЛЕНИЯ МЕХАТРОННЫМИ И РОБОТОТЕХНИЧЕСКИМИ СИСТЕМАМИ НА ПРИМЕРЕ КОПТЕРА TELLO

Цель работы – получить знания и умения разработки программного обеспечения для управления мехатронных и робототехнических систем по заранее заданному алгоритму; получить умение разработки программно-алгоритмического обеспечения для управления мехатронными и робототехническими системами по информации с датчиков.

Методические указания

В ходе выполнения практической работы необходимо решить следующие задачи:

– изучить SDK на квадракоптер tello в части команд подключения дрона к сети, управления, считывания показаний датчиков;

– написать код для получения взлета/посадки коптера tello, полета по заданной траектории, считывания информации с датчиков; провести тестирование кода.

Для разработки программного обеспечения нам потребуется библиотека djitellopy2, основанная на использовании официальной версии Tello SDK. Для ее установки используйте команду pip install djitellopy. Установленная библиотека поддерживает основные команды управления дроном, получение видеопотока, получение информации с датчиков. После установки этой библиотеки вы можете воспользоваться ее методами, часть из которых мы рассмотрим на примере программы, приведенной в листинге 1. Здесь осуществляется подключение к дрону командой connect(), в словарь commands вносятся команды перемещения, после чего дается команда takeoff() на взлет, после взлета коптер ожидает 5 с и ему дается команда move на перемещение назад на 50 см, после паузы длительностью 5 с дается команда развернуться по часовой стрелке на 90° и команда на посадку.

Листинг 1

```
from djitellopy import Tello
import time
tello =Tello()
tello.connect()
commands={1:'forward   ',   -1:'back',   2:'up',   -2:'down',
3:'right',-3:'left', 4:'tello.rotate_clockwise(90)'}
tello.takeoff()
time.sleep(5)
tello.move(commands[-1],50)
time.sleep(5)
exec(commands[4])
tello.land()
```

Запустите программу и проведите ее тестирование. Вы можете убедиться, что после подачи команды коптер высылает статус ее выполнения. В случае, если команда не пришла, она отправляется вновь. Обратите внимание, что если коптер не получает команду в течение 15 с, то он автоматически приземляется. В качестве самостоятельной работы модифицируйте код таким образом, чтобы коптер висел в воздухе в течение заданного времени. Запрограммируйте траекторию полета коптера. Получите информацию с датчиков о высоте полета и уровне заряда батареи. Для получения требуемой информации воспользуйтесь командами get_height(), get_battery(). Напишите программу, обеспечивающую посадку коптера при достижении заданного уровня заряда батареи или сигнала Wi-Fi.

ПОЛУЧЕНИЕ ИНФОРМАЦИИ С КАМЕР В ВИДИМОМ ДИАПАЗОНЕ СПЕКТРА ИЗЛУЧЕНИЯ И ЕЕ ИСПОЛЬЗОВАНИЕ ДЛЯ РЕШЕНИЯ ЗАДАЧ В ОБЛАСТИ УПРАВЛЕНИЯ МЕХАТРОННЫМИ И РОБОТОТЕХНИЧЕСКИМИ СИСТЕМАМИ НА ПРИМЕРЕ КОПТЕРА TELLO

Цель работы – получить знания и навыки захвата изображений с камер в видимом диапазоне и использования полученной информации при разработке программно-алгоритмического обеспечения для управления мехатронными и робототехническими системами.

Методические указания

В ходе выполнения практической работы необходимо решить следующие задачи:

– изучить SDK на квадракоптер tello в части команд подключения дрона к сети и получения видеопотока;

– написать код для получения видеопотока с коптера tello и записи видео в видеофайл, провести тестирование кода.

При использовании в составе мехатронных систем web-камер для захвата изображений с них и последующей обработки полученной информации удобно использовать язык программирования python и библиотеку opencv. Пример программы, выполняющей функцию захвата изображения и вывода его в графическое окно, представлен в листинге 1.

Листинг 1

```
import numpy as np
import cv2
cap = cv2.VideoCapture(0)
fourcc = cv2.VideoWriter_fourcc(*'XVID')
out = cv2.VideoWriter('output.avi',fourcc, 20.0, (640,480))
while(cap.isOpened()):
    ret,frame = cap.read()
    if ret==True:
        gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
```

```
            cv2.imshow('frame',gray)
            out.write(frame)
            if cv2.waitKey(1) & 0xFF == ord('q'):
                break
        else:
            break
    cap.release()
    out.release()
    cv2.destroyAllWindows()
```

Для захвата изображения вначале необходимо создать видеообъект, используя функцию **VideoCapture()**, аргументом которой является номер камеры или имя файла, содержащего видео. Иногда видеообъект не создается (например, камера или видеофайл не найдены). В этом случае генерируется ошибка, проверить которую можно используя функцию **isOpened()**. Если эта функция возвращает True, то все в порядке, и можно считывать кадры. После того, как создан видеообъект, можно кадр за кадром считывать изображение. Функция **read()** возвращает True, если кадр считан успешно, и False – если произошла ошибка. Таким образом, если кадр считан успешно, он помещается в графическое окно. Этот процесс будет осуществляться до тех пор, пока не будет нажата клавиша с буквой ‘q’. В конце программы не забудьте освободить память, занятую видео, используя функцию **release().**

Изображение можно вывести в различных цветовых оттенках. Для этого необходимо использовать функцию **cvtColor()**, первым аргументом которой является считанный кадр, а вторым – флаг, определяющий формат вывода изображения. Например, для вывода кадра в оттенках серого можно использовать следующий код:

gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY).

Захваченное с камеры изображение можно записывать в файл. Для этого необходимо определить тип кодека, используя 4-байтный код FourCC. Тип кодека зависит от используемой платформы. Можно использовать следующие типы: DIVX, XVID, MJPG, X264, WMV1, WMV2. Далее вызывается функция **VideoWriter()**, аргументами которой являются имя записываемого файла, тип кодека, количество кадров в секунду и размер кадров. В качестве последнего

аргумента можно указать значение флага, определяющего цветовые оттенки изображения.

Рассмотрим пример захвата изображения с камеры квадракоптера dji tello. Предварительно необходимо установить библиотеку djitellopy2. Эта библиотека использует официальную SDK и поддерживает функции реализации всех команд tello, позволяет легко извлекать видеопоток, получать и анализировать пакеты состояния, управлять роем дронов. После установки соответствующей библиотеки ее можно импортировать в основном коде, для этого используйте команду **from** djitellopy **import** Tello. После чего мы можем создать объект – tello, используя команду tello =Tello(). И подключиться к нашему дрону по сети Wi-Fi, используя tello.connect(). Не забудьте перед подключением войти в сеть tello на рабочем компьютере, с которого будет осуществляться управление коптером (базовой станции). Далее следует запустить функцию получения видеопотока tello.streamon() и в цикле считывать кадры frame1=tello.get_frame_read(), выводя их в графическое окно. Полный код программы для захвата изображения представлен в листинге 2.

Листинг 2

```
import cv2
from djitellopy import Tello
import  sys

tello =Tello()
tello.connect()
tello.streamon()

while True:
    frame1=tello.get_frame_read()
    img=frame1.frame
    cv2.imshow("tello camera",img)
    cv2.waitKey(1)
    if cv2.waitKey(1) & 0xFF == ord('q'):
        tello.streamoff()
        break
```

Практическая работа 4

ПОЛУЧЕНИЕ ИНФОРМАЦИИ С КАМЕР В ИНФРАКРАСНОМ ДИАПАЗОНЕ СПЕКТРА ИЗЛУЧЕНИЯ И ЕЕ ИСПОЛЬЗОВАНИЕ ДЛЯ РЕШЕНИЯ ПРАКТИЧЕСКИХ ЗАДАЧ

Цель работы – получить навыки захвата изображений с тепловизионных камер и обработки изображений в инфракрасном диапазоне спектра излучения с использованием программных средств OpenCV; овладеть навыком организации работ по разработке алгоритмов и программ для управления средствами измерительной техники на основе оптических методов контроля (на примере теплового бесконтактного метода контроля).

Методические указания

Тепловизионная камера (тепловизор), установленная на дроне, превращает его в мощный инструмент, который может использоваться во многих секторах: строительстве, горнодобывающей промышленности, электроэнергетике, обследованиях, пожаротушении, поисковых и спасательных операциях. Дроны, оснащенные тепловизорами, имеют множество применений, обнаруживая тепло, исходящее почти от всех объектов и материалов, превращая их в изображения и видео. В данной работе будем использовать тепловизор фирмы Fluke, с помощью которого мы получим изображения объектов контроля в инфракрасном диапазоне спектра излучения и проведем их анализ.

Порядок выполнения работы следующий.

Включите тепловизор.

Запишите видеоряд для интересующего объекта контроля (это может быть трубопровод, линия электропередачи, технологическое оборудование и т.п.) или получите ряд видеоснимков (например, рис. 1 – 3).

Проведите анализ полученных изображений видеоряда. Отметьте особенности объектов контроля, по которым их можно идентифицировать. Например, если ставится задача идентификации мест утечки теплоносителя из трубы, то в указанных местах температура должна быть выше. Если стоит задача иден-

тификации плохих контактов проводников под напряжением, то в месте плохого контакта происходит нагрев, что также должно быть видно на инфракрасных изображениях.

Рис. 1. Инфракрасное изображение автомобиля

Рис. 2. Инфракрасное изображение фрагмента улицы

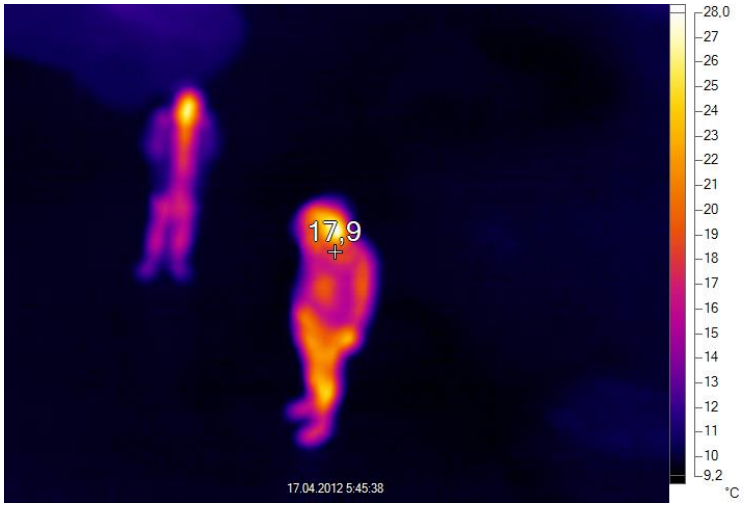


Рис. 3. Инфракрасное изображение людей с высоты 5 м

После того, как вы выделите особенности, характерные для вашего объекта контроля, разработайте алгоритм и программу для идентификации объектов.

Рассмотрим пример. Пусть стоит задача определения контуров людей и автомобилей в темное время суток. В результате анализа изображений, приведенных на рис. 1 – 3, мы обнаружим, что основными источниками тепла являются автомобили и люди. При этом контуры людей и машин имеют существенные отличия. Применим к изображениям 1 и 3 операции порогового преобразования. Получим следующие изображения (рис. 4, 5).

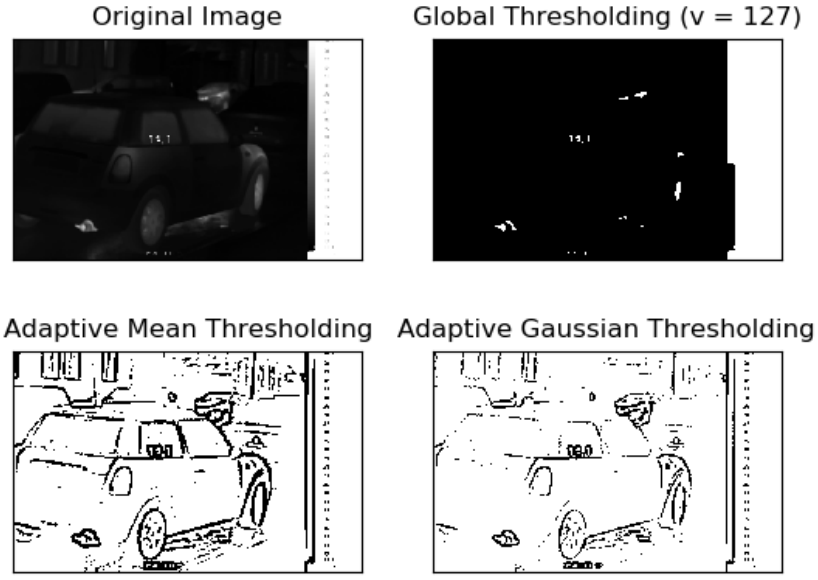


Рис. 4. Изображение автомобиля после порогового преобразования

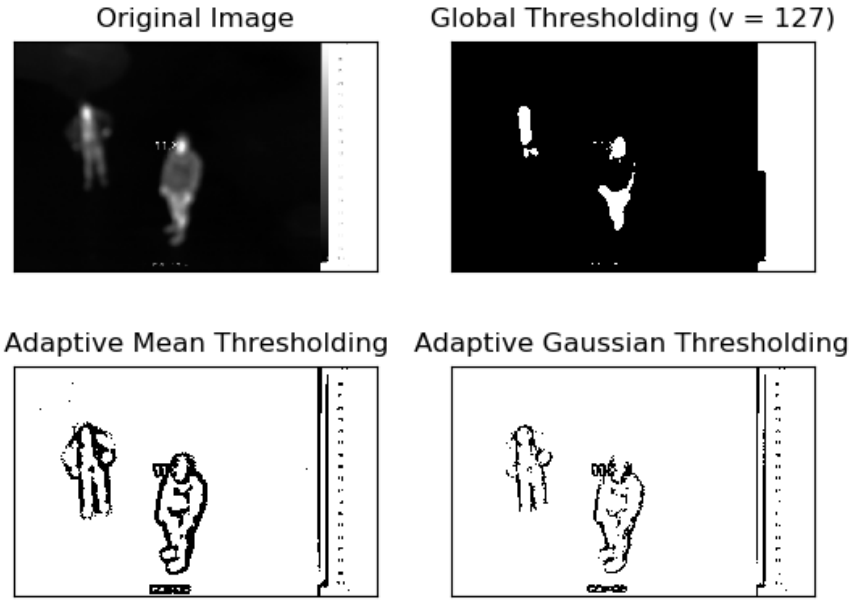


Рис. 5. Изображение людей после порогового преобразования

Для получения представленных изображений мы использовали нижеприведенный код, в основе которого использованы стандартные функции глобального порогового и адаптивного порогового преобразования.

Листинг 1

```
import cv2
import numpy as np
from matplotlib import pyplot as plt
img = cv2.imread('term3.bmp',0)
img = cv2.medianBlur(img,5)
ret,th1 = cv2.threshold(img,90,255,cv2.THRESH_BINARY)
th2 =
cv2.adaptiveThreshold(img,255,cv2.ADAPTIVE_THRESH_MEAN_C,\
                              cv2.THRESH_BINARY,15,2)
th3 =
cv2.adaptiveThreshold(img,255,cv2.ADAPTIVE_THRESH_GAUSSIAN_C,\
                              cv2.THRESH_BINARY,15,2)

cv2.imshow('Map',th2)
cv2.waitKey(0)
titles = ['Original Image', 'Global Thresholding (v = 127)',
'Adaptive Mean Thresholding', 'Adaptive Gaussian Thresholding']
images = [img, th1, th2, th3]
for i in range(4):
    plt.subplot(2,2,i+1),plt.imshow(images[i],'gray')
    plt.title(titles[i])
    plt.xticks([]),plt.yticks([])
plt.show()
```

Далее мы можем определить и нарисовать контуры найденных объектов (рис. 6), используя код

```
img1 = cv2.imread('term3.bmp',1)
…..
im = cv2.drawContours(img1, contours, -1, (0,255,0), 1)
cv2.imshow('Map',img1)
```

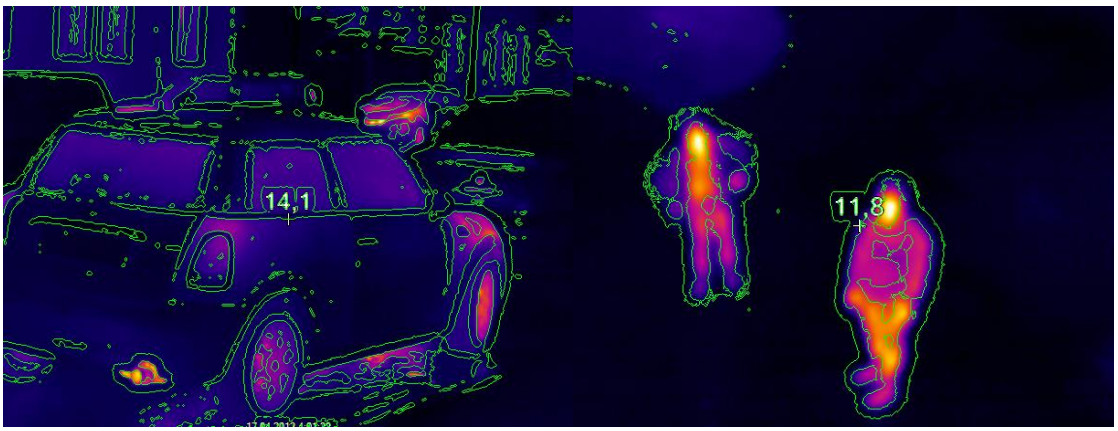


Рис. 6. Контуры найденных объектов

На последнем этапе нам необходимо нарисовать рамку, обводящую контуры людей и автомобилей. Проанализируйте имеющиеся контуры и предложите алгоритм решения этой задачи.

Организуйте команду разработчиков программного обеспечения для теплового оптического контроля оборудования (например, отопительного оборудования). Распределите роли (кто отвечает за проведение измерений, получение тепловизионных кадров, обработку изображений, написание программы и т.д.). Выполните разработку.

Практическая работа 5

МУЛЬТИПРОЦЕССОРНАЯ ОБРАБОТКА ДАННЫХ ПРИ РЕШЕНИИ ЗАДАЧ В ОБЛАСТИ УПРАВЛЕНИЯ МЕХАТРОННЫХМИ И РОБОТОТЕХНИЧЕСКИХМИ СИСТЕМАМИ НА ПРИМЕРЕ КОПТЕРА TELLO

Цель работы – получить знания и навыки мультипроцессорной обработки данных при управлении мехатронными системами; получить навыки организации работ по разработке алгоритмов и программ для управления летающими робототехническими средствами.

Методические указания

Мультипроцессорная обработка позволяет параллельно выполнять ряд процессов, что является необходимым при решении задач управления мехатронными системами. Например, при управлении полетом квадракоптера мы используем информацию, получаемую с его камеры, на основе которой система управления задает параметры полета. В этом случае речь идет о двух параллельных процессах – получении и обработки информации с камеры, управлении полетом на основе данной информации. В языке python имеется модуль multiprocessing , поддерживающий порождение процессов с использованием API. Рассмотрим основные его особенности и принципы работы.

Одним из объектов модуля является объект Pool, позволяющий распараллеливать выполнение функций по входным значениям. С помощью него можно создать пул процессов, которые будут выполнять переданные в него задачи. Синтаксис следующий:

class multiprocessing.pool.**Pool**([*processes*[, *initializer*[, *initargs*[, *maxtasksperchild*[, *context*]]]]]), где *processes* – количество используемых рабочих процессов; если *initializer* не None, то каждый рабочий процесс при запуске будет вызывать initializer(*initargs); *maxtasksperchild* – это количество задач, которые рабочий процесс может выполнить до того, как он завершится и будет заменен новым рабочим процессом, чтобы освободить неиспользуемые ресурсы. По молчанию *maxtasksperchild* – None, что означает, что рабочие процессы будут жить столько же, сколько и пул.

context может использоваться для указания контекста, используемого для запуска рабочих процессов. Обычно пул создается с помощью функции multiprocessing.Pool() или метода Pool() объекта контекста. Например, нам необходимо параллельно вычислить произведение ряда чисел на число 10. Для этого можно создать функцию func1(), выполняющую данную операцию и в основной части программы создать пул процессов.

Листинг 1

```
from multiprocessing import Pool
def func1(x):
    return x*10

if __name__ == '__main__':
    with Pool(2) as p:
        print(p.map(func1, [1, 2, 3]))
```

В multiprocessing порождение процессов осуществляется с помощью объекта Process и последующего вызова его метода start(). Это означает, что определенные действия будут выполняться в отдельном процессе. Например, в качестве таких действий может быть просто вывод строки.

Листинг 2

```
from multiprocessing import Process

def func(name):
    print('Privet', name)

if __name__ == '__main__':
    p = Process(target=func, args=('Pavel',))
    p.start()
    p.join()
```

Синтаксис объекта следующий:

class multiprocessing.**Process**(*group=None*, *target=None*, *name=None*, *args=()*, *kwargs={}*, *, *daemon=None*), где *group* всегда должен быть None; *target* – вызываемый объект, в нашем случае это функция func; *name* – имя процесса; *args* – кортеж аргументов для целевого вызова; *kwargs* – словарь ключевых аргументов для целевого вызова. Если указан аргумент *daemon*, содержащий

только ключевой аргумент, для флага процесса daemon устанавливается значение True или False. Если None (по умолчанию), этот флаг будет унаследован от процесса создания. Для завершения процесса используется метод join().

Рассмотрим пример создания двух параллельных процессов – основного и дочернего. В основном процессе у нас выполняется задача, записанная в функции info, при этом параллельно мы запускаем процесс, в котором выполняется задача в функции func.

Листинг 3

```
from multiprocessing import Process
import os

def info(title):
    print(title)
    print('module name:', __name__)
    print('parent process:', os.getppid())
    print('process id:', os.getpid())

def func(name):
    info('function f')
    print('Privet', name)

if __name__ == '__main__':
    info('main line')
    p = Process(target=func, args=('Pavel',))
    p.start()
    p.join()
```

Созданные процессы могут обмениваться данными. Для этого можно использовать функцию Pipe(), которая возвращает пару объектов соединения, соединенных конвейером (pipe), который по умолчанию является дуплексным (двусторонним). Два объекта подключения, возвращаемые Pipe(), представляют собой два конца конвейера. Каждый объект подключения содержит методы send() и recv(), предназначенные для записи и чтения данных. В приведенном ниже примере создаются два объекта parent_conn, child_conn. Объект child_conn отправляет строку объекту parent_conn.

Листинг 4

```
from multiprocessing import Process, Pipe

def f(conn):
    conn.send([42, None, 'hello'])
    conn.close()

if __name__ == '__main__':
    parent_conn, child_conn = Pipe()
    p = Process(target=f, args=(child_conn,))
    p.start()
    print(parent_conn.recv())   # распечатает "[42, None,
'hello']"
    p.join()
```

Обратите внимание, что данные в конвейере могут быть повреждены, если два процесса (или потока) попытаются одновременно прочитать или записать на конец конвейера одно и то же.

Так же при управлении мехатронной системой с некоторой базовой станции, например компьютера, бывает необходимо обеспечить оператору возможность вмешаться в процесс, например экстренно посадить дрон или остановить движущийся мобильный робот. Для этого полезным будет модуль signal. Это средство, позволяющее получить сигнал от операционной системы базовой станции и обработать его. Когда операционная система получает события, например нажатие клавиш, она может их передавать в программу в виде сигналов. Например, при нажатии комбинации Ctrl + C операционная система генерирует сигнал SIGNT и передает его программе. Обработав этот сигнал, мы можем выполнить те или иные действия, например дать команду дрону на посадку. В приведенном ниже примере мы прерываем вывод строки в бесконечном цикле после нажатия комбинации клавиш Ctrl + C.

Листинг 5

```
import signal
import time

def exit_handler(signum, frame):
    print('Exiting....')
    exit(0)
```

```
signal.signal(signal.SIGINT, exit_handler)

while 1:
    print("Press Ctrl + C")
    time.sleep(3)
```

Теперь применим полученную информацию для управления квадракоптером. Обеспечим его взлет, вращение на месте и получение визуальной информации с камеры, которую будем выводить в графическом окне. Разделим обработку данных на два процесса – в первом процессе будет осуществляться считывание кадров с камеры и управление полетом (в простейшем случае вращение вокруг оси), во-втором процессе – вывод информации в графическое окно. Первый процесс осуществляется функцией movement, в которой создается объект – коптер tello, обеспечивается взлет коптера tello.takeoff(), подключение камеры с последующим считыванием кадров tello.connect(), tello.streamon(), frame_read = tello.get_frame_read(), а также вращение tello.send_command_without_return('cw 90'). Второй процесс осуществляется функцией show_video, которая получает кадры с камеры и выводит их в графическое окно.

Листинг 6

```
import signal
import time,cv2
from multiprocessing import Manager, Process, Pipe, Event
from djitellopy import Tello

tello = None

def signal_handler(sig):
    print("Signal Handler")
    if tello:
        try:
            tello.streamoff()
            tello.land()
        except:
            pass

    sys.exit()
```

```
    def movement(exit_event, show_video_conn,  run_pid,
track_face, fly):

        signal.signal(signal.SIGINT, signal_handler)

        global tello
        tello = Tello()

        tello.connect()

        tello.streamon()
        frame_read = tello.get_frame_read()

        if fly:
            tello.takeoff()


        while not exit_event.is_set():

            frame = frame_read.frame
            show_video_conn.send(frame)
            tello.send_command_without_return('cw 90')

        signal_handler(None)

    def show_video(exit_event, pipe_conn):
        signal.signal(signal.SIGINT, signal_handler)

        while True:
            frame = pipe_conn.recv()
            cv2.imshow("Drone Face Tracking", frame)
            cv2.waitKey(1)
            if cv2.waitKey(1) & 0xFF == ord('q'):
                exit_event.set()
    if __name__ == '__main__':

        signal.signal(signal.SIGINT, signal_handler)

        run_pid = True
        track_face = True
        fly = True

        parent_conn, child_conn = Pipe()

        exit_event = Event()
```

```
        with Manager() as manager:
            p1 = Process(target=movement,
                         args=(exit_event, child_conn, run_pid,
track_face, fly, ))

            p2 = Process(target=show_video, args=(exit_event,
parent_conn,))

            p2.start()
            p1.start()
            p1.join()
            p2.terminate()
            p2.join()
```

Протестируем данную программу. Мы видим, что основная проблема, связанная с зависанием изображения с камеры при полете, решена. Однако, команда tello.send_command_without_return('**cw 90**') непрерывно передается коптеру, что явно избыточно. Помимо этого, как мы можем увидеть, она не всегда исполняется, поскольку параллельно осуществляется второй процесс – считывание кадров. Попробуем решить эту проблему, выполняя указанные функции параллельно.

МНОГОПОТОЧНАЯ ОБРАБОТКА ДАННЫХ ПРИ РЕШЕНИИ ЗАДАЧ В ОБЛАСТИ УПРАВЛЕНИЯ МЕХАТРОННЫМИ И РОБОТОТЕХНИЧЕСКИМИ СИСТЕМАМИ НА ПРИМЕРЕ КОПТЕРА TELLO

Цель работы – получить знания и навыки многопоточной обработки данных при управлении мехатронными системами

Методические указания

Многопоточность позволяет процессору как бы одновременно обрабатывать несколько задач внутри одного процесса. Процесс, выполняемый ядром процессора, можно представить в виде одного или нескольких потоков, которые разделяют выделенную ядру процессора память и процессорное время. На самом деле процессы выполняются последовательно, но переключение между ними происходит быстро, поэтому создается впечатление, что они работают параллельно. В нашем случае выделим в нашем процессе *movement* поток, в котором коптеру с интервалом времени 1 с будет даваться команда поворота на 90°. Для этого внутри функции *movement* создадим функцию control():

```
def control():
    while True:
        tello.send_command_without_return('cw 90')
        time.sleep(1)
```

Запустим поток, в котором будет исполняться созданная функция:

```
th=Thread(target=control)
th.start()
```

Полный листинг получившейся программы представлен ниже.

```
import signal
import time,cv2
from multiprocessing import Manager, Process, Pipe, Event
from djitellopy import Tello
```

```
    from threading import Thread

    tello = None

    def signal_handler(sig):
        print("Signal Handler")

        if tello:
            try:
                tello.streamoff()
                tello.land()
            except:
                pass
        sys.exit()

    def movement(exit_event, show_video_conn,  run_pid,
track_face, fly):

        signal.signal(signal.SIGINT, signal_handler)
        global tello
        tello = Tello()
        tello.connect()
        tello.streamon()
        frame_read = tello.get_frame_read()

        if fly:
            tello.takeoff()

        def control():
            while True:
                tello.send_command_without_return('cw 90')
                time.sleep(1)

        th=Thread(target=control)
        th.start()

        while not exit_event.is_set():

            frame = frame_read.frame
            show_video_conn.send(frame)
        signal_handler(None)

    def show_video(exit_event, pipe_conn):
        signal.signal(signal.SIGINT, signal_handler)
```

```
        while True:
            frame = pipe_conn.recv()
            # display the frame to the screen
            cv2.imshow("Drone Face Tracking", frame)
            cv2.waitKey(1)
            if cv2.waitKey(1) & 0xFF == ord('q'):
                exit_event.set()

    if __name__ == '__main__':

        signal.signal(signal.SIGINT, signal_handler)

        run_pid = True

        track_face = True
        fly = True

        parent_conn, child_conn = Pipe()

        exit_event = Event()

        with Manager() as manager:
            p1 = Process(target=movement,
                         args=(exit_event, child_conn, run_pid,
track_face, fly, ))

            p2 = Process(target=show_video, args=(exit_event,
parent_conn,))
            p2.start()
            p1.start()
            p1.join()
            p2.terminate()
            p2.join()
```

Практическая работа 7

ДЕТЕКТИРОВАНИЕ ОБЪЕКТОВ В ВИДЕОПОТОКЕ С ИСПОЛЬЗОВАНИЕМ OPENCV DNN

Цель работы – получить знания и умения использовать информацию с камер в видимом диапазоне спектра излучения для решения задачи определения объектов на изображениях и управления мехатронными и робототехническими системами.

Методические указания

Область науки, занимающаяся разработкой алгоритмов компьютерного зрения, существует с конца 1960-х годов. Задачи классификации изображений и обнаружения объектов являются одними из старейших в области компьютерного зрения, которые исследователи пытались решить в течение многих десятилетий. Используя нейронные сети и глубокое обучение, мы достигли стадии, когда компьютеры могут начать действительно понимать и распознавать объект с высокой точностью, во многих случаях даже превосходя людей. В данной работе мы будем использовать модуль DNN OpenCV, предназначенный для работы с нейронными сетями и методами глубокого обучения. Благодаря высоко оптимизированной производительности процессора новички также могут легко приступить к работе, даже если у них нет очень мощной системы с поддержкой графического процессора.

Мы все знаем OpenCV как одну из лучших библиотек компьютерного зрения. Модуль OpenCV DNN поддерживает только вывод глубокого обучения на изображениях и видео. Он не поддерживает точную настройку и обучение. Тем не менее модуль OpenCV DNN может послужить идеальной отправной точкой для любого начинающего разработчика систем технического зрения.

Одним из преимуществ модуля OpenCV DNN является то, что он оптимизирован для процессоров Intel. Мы можем создавать быстродействующие приложения для распознавания объектов на видеопотоке в реальном времени. С помощью модуля мы может выполнять следующие функции: классификация изображений, обнаружение объектов, сегментация изображений, обнаружение и распознавание текста, оценка позы, оценка глубины, проверка и обнаружение личности и лица.

Мы будем использовать предварительно подготовленные модели. Эти модели были обучены на базе набора данных MS COCO, текущего базового набора данных для моделей обнаружения объектов на основе глубокого обучения.

Набор данных MS COCO содержит 80 классов объектов, начиная от человека, автомобиля и заканчивая зубной щеткой. Мы также будем использовать текстовый файл для загрузки всех меток, присутствующих в наборе данных MS COCO, для обнаружения объектов.

Мы будем использовать детектор ***MobileNet SSD***, который был обучен набору данных MS COCO с использованием платформы глубокого обучения TensorFlow. Модели SSD, как правило, работают быстрее по сравнению с другими моделями обнаружения объектов. Кроме того, ***MobileNet SSD*** также делает их менее трудоемкими. Таким образом, это хорошая модель для начала изучения обнаружения объектов с помощью OpenCV DNN.

Для начала импортируем модули cv2, time, numpy:

```
import cv2
import time
import numpy as np
```

Далее нам необходимо загрузить файл класса MS COCO:

object_detection_classes_coco.txt.

```
with open('object_detection_classes_coco.txt', 'r') as f:
    class_names = f.read().split('\n')
```

Он содержит имена классов, расположенные в одну строку. Наряду с этим мы создадим массив ЦВЕТОВ, содержащий кортежи из трех целых значений. Это случайные цвета, которые мы можем применить при рисовании ограничительной рамки для каждого класса. У нас будет ограничивающая рамка разного цвета для каждого класса, и нам будет довольно легко различать классы в конечном результате.

```
COLORS = np.random.uniform(0, 255, size=(len(class_names), 3))
```

Далее мы загрузим модель SSD Mobile Net с помощью функции readint().

```
model = cv2.dnn.readNet(model='frozen_inference_graph.pb',
                        config='ssd_mobilenet_v2_coco_2018_03_29.pbtxt.txt',
                        framework='TensorFlow')
```

В приведенном выше блоке кода:

– аргумент модели принимает в качестве входных данных путь к файлу **frozen_inference_graph.pb** предварительно обученной модели, содержащей веса;

– аргумент config принимает путь к файлу конфигурации модели, который является текстовым файлом protobuf.

Наконец, мы определяем структуру, которая в данном случае является тензорным потоком.

Далее мы прочитаем кадр из видеопотока. Для этого создадим объект VideoCapture() для получения видеопотока и объект VideoWriter() – для правильного сохранения полученных видеокадров.

```
cap = cv2.VideoCapture('video_1.mp4')
frame_width = int(cap.get(3))
frame_height = int(cap.get(4))
out = cv2.VideoWriter('video_result.mp4',
cv2.VideoWriter_fourcc(*'mp4v'), 30, (frame_width, frame_height))
```

На данный момент мы готовы считывать видеопотоки и применять модель для распознавания объектов. Следующий шаг состоит в том, чтобы считывать кадр и выполнять обнаружение объектов в каждом кадре.

```
while cap.isOpened():
    ret, frame = cap.read()
    if ret:
        image = frame
        image_height, image_width, _ = image.shape
        # создаем объект для изображения
        blob = cv2.dnn.blobFromImage(image=image, size=(300,
300), mean=(104, 117, 123),
                                     swapRB=True)
        # запускаем таймер для расчета  FPS
        start = time.time()
        model.setInput(blob)
        output = model.forward()
        # останавливаем таймер после детектирования объекта
        end = time.time()
        # вычисляем FPS для текущего кадра
        fps = 1 / (end-start)
        # проходим по каждому обнаруженному объекту
```

```
            for detection in output[0, 0, :, :]:
                # извлекаем точность обнаружения
                confidence = detection[2]
                # рисуем прямоугольник, обводящий обнаруженный
объект                 if confidence > .4:
                    # определяем id класса
                    class_id = detection[1]
                    class_name = class_names[int(class_id)-1]
                    color = COLORS[int(class_id)]
                    # вычисляем координаты рамки
                    box_x = detection[3] * image_width
                    box_y = detection[4] * image_height
                    # вычисляем ширину и высоту рамки
                    box_width = detection[5] * image_width
                    box_height = detection[6] * image_height
                    # рисуем рамку для каждого объекта
                    cv2.rectangle(image, (int(box_x),
int(box_y)), (int(box_width), int(box_height)), color, thick-
ness=2)
                    # выводим класс объекта
                    cv2.putText(image, class_name, (int(box_x),
int(box_y - 5)), cv2.FONT_HERSHEY_SIMPLEX, 1, color, 2)
                    # помещаем FPS в кадр
                    cv2.putText(image, f"{fps:.2f} FPS", (20,
30), cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 255, 0), 2)

            cv2.imshow('image', image)
            out.write(image)
            if cv2.waitKey(10) & 0xFF == ord('q'):
                break
        else:
            break

    cap.release()
    cv2.destroyAllWindows()
```

Модифицируйте данную программу, обеспечив детектирование видеопотока с камеры коптера tello. Протестируйте программу.

ОБЕСПЕЧЕНИЕ ВЗАИМОДЕЙСТВИЯ МЕЖДУ ПРОЦЕССАМИ ОБРАБОТКИ ДАННЫХ

Цель работы – получить умения и навыки организации мультипроцессорной обработки данных.

Методические указания

Рассмотрим следующую структуру программного обеспечения для управления коптером (рис. 1).

Параллельно выполняются два процесса – ***movement*** и ***show_video.*** Первый процесс обеспечивает полет коптера и считывание кадров frames, которые передаются во второй процесс, где происходит анализ этих кадров и поиск заданных объектов на них с использованием нейросети dnn. Внутри первого процесса выполняются два потока ***control*** и ***box***. В потоке ***control*** обеспечивается зависание коптера и поворот с интервалом 1 по часовой стрелке на 30°. Процесс ***box*** получает и выводит значение ширины width рамки, ограничивающей найденный объект.

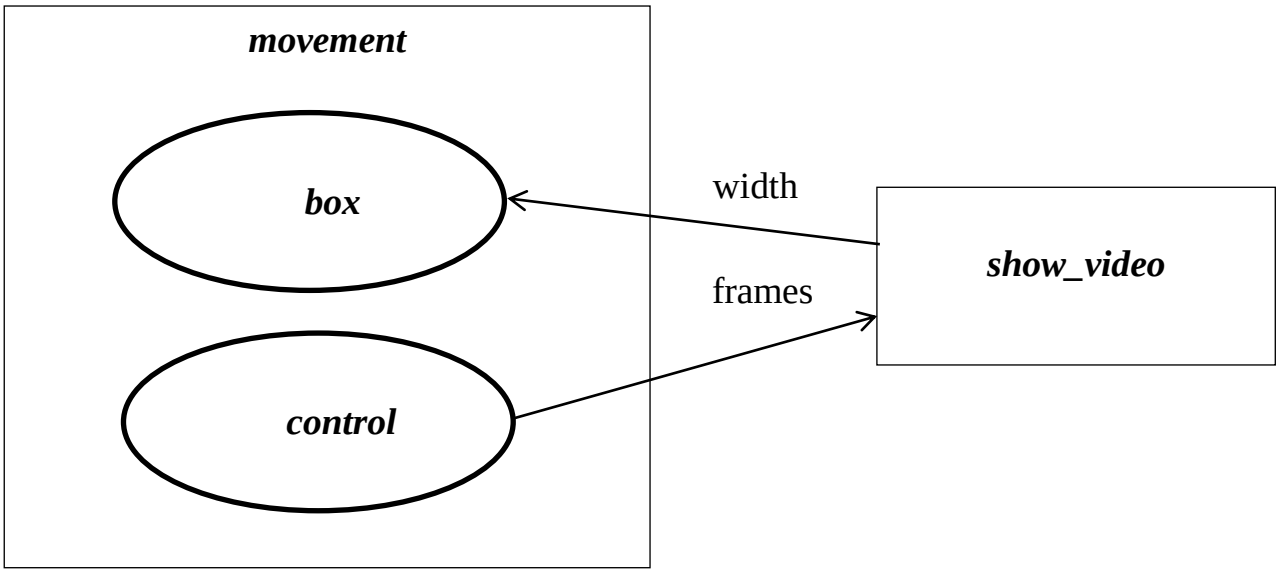


Рис. 1. Структура программы управления коптером

Для создания канала взаимодействия между процессами используем код

```
#канал для передачи кадров из процесса 1 в процесс 2
parent_conn, child_conn = Pipe()
#канал для передачи размера найденной метки из процесса 2
в процесс 1
parent_conn2, child_conn2 = Pipe()
```

Для создания и запуска процессов используется код

```
p1 = Process(target=movement,
             args=(exit_event, child_conn, fly, parent_conn2, ))

p2 = Process(target=show_video, args=(exit_event, par-
ent_conn, child_conn2, ))

p2.start()
p1.start()
```

Для запуска потоков внутри процесса ***movement*** используется код

```
th=Thread(target=control)
th.start()
th2=Thread(target=box)
th2.start()
```

Для передачи кадров frames в процесс ***show_video*** используется код

```
show_video_conn.send(frame)
```

Для приема кадров frames в процессе ***show_video*** используется код

```
image = pipe_conn.recv()
```

Для передачи параметра width в процесс ***movement*** используется код

```
movement_conn.send(box_width)
```

Для приема параметра width в процессе ***movement*** используется код

```
w = show_video_conn2.recv()
```

Таким образом, полученная структура параллельного выполнения процессов и потоков обеспечивает большую производительность программы и снижает «подвисание» при передаче кадров и управляющих команд на коптер. Полный код программы приведен в листинге ниже.

```
    import signal
    import time,cv2
    from multiprocessing import Manager, Process, Pipe, Event
    from djitellopy import Tello
    from threading import Thread
    import numpy as np
    import time

    tello = None

    def movement(exit_event, show_video_conn, fly,
show_video_conn2):

        global tello
        tello = Tello()
        tello.connect()
        tello.streamon()
        frame_read = tello.get_frame_read()

        if fly:
            tello.takeoff()

        def control():
            while True:
                tello.send_command_without_return('cw 30')
                time.sleep(1)
        def box():

            w = show_video_conn2.recv()
            print(w)
            time.sleep(1)

        th=Thread(target=control)
        th.start()
        th2=Thread(target=box)
        th2.start()

        while not exit_event.is_set():

            frame = frame_read.frame
            show_video_conn.send(frame)

        signal_handler(None)
```

```
    def show_video(exit_event, pipe_conn, movement_conn):

        with open('object_detection_classes_coco.txt', 'r') as f:
            class_names = f.read().split('\n')
    # get a different color array for each of the classes
        COLORS = np.random.uniform(0, 255,
size=(len(class_names), 3))

    # load the DNN model
        model =
cv2.dnn.readNet(model='frozen_inference_graph.pb',
                                con-
fig='ssd_mobilenet_v2_coco_2018_03_29.pbtxt.txt',
                                framework='TensorFlow')

        while True:
            image = pipe_conn.recv()
            image_height, image_width, _ = image.shape
            blob = cv2.dnn.blobFromImage(image=image, size=(300,
300), mean=(104, 117, 123),
                                         swapRB=True)
            # start time to calculate FPS
            start = time.time()
            model.setInput(blob)
            output = model.forward()
            # end time after detection
            end = time.time()
            # calculate the FPS for current frame detection
            fps = 1 / (end-start)

            for detection in output[0, 0, :, :]:
                # extract the confidence of the detection
                confidence = detection[2]
                # draw bounding boxes only if the detection con-
fidence is above...
                # ... a certain threshold, else skip
                class_id = detection[1]
                if confidence > .4 and class_id==1:
                    # get the class id
```

```
                    # map the class id to the class
                    class_name = class_names[int(class_id)-1]
                    color = COLORS[int(class_id)]
                    # get the bounding box coordinates
                    box_x = detection[3] * image_width
                    box_y = detection[4] * image_height
                    # get the bounding box width and height
                    box_width = detection[5] * image_width
                    box_height = detection[6] * image_height
                    #отправляем размер рамки в процесс 1
                    movement_conn.send(box_width)
                    # draw a rectangle around each detected ob-
ject
                    cv2.rectangle(image, (int(box_x),
int(box_y)), (int(box_width), int(box_height)), color, thick-
ness=2)
                    # put the class name text on the detected ob-
ject
                    cv2.putText(image, class_name, (int(box_x),
int(box_y - 5)), cv2.FONT_HERSHEY_SIMPLEX, 1, color, 2)
                    # put the FPS text on top of the frame
                    cv2.putText(image, f"{fps:.2f} FPS", (20,
30), cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 255, 0), 2)

            # display the frame to the screen

            cv2.imshow("Drone Tracking", image)

            if cv2.waitKey(1) & 0xFF == ord('q'):
                exit_event.set()

    if __name__ == '__main__':

        fly = True

        #канал для передачи кадров из процесса 1 в процесс 2
        parent_conn, child_conn = Pipe()
        #канал для передачи размера найденной метки из процесса 2
в процесс 1
        parent_conn2, child_conn2 = Pipe()
```

```
        exit_event = Event()

        with Manager() as manager:
            p1 = Process(target=movement,
                         args=(exit_event, child_conn, fly, par-
ent_conn2, ))

            p2 = Process(target=show_video, args=(exit_event,
parent_conn, child_conn2, ))

            p2.start()
            p1.start()
            p1.join()
            p2.terminate()
            p2.join()
```

Практическая работа 9

ТЕСТИРОВАНИЕ И РАСЧЕТ НАДЕЖНОСТИ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ МЕХАТРОННЫХ И РОБОТОТЕХНИЧЕСКИХ СИСТЕМ

Цель работы – получить навыки применения программных средств Matlab или MathCad для расчета надежности программного обеспечения мехатронных и робототехнических систем, средств автоматики, измерительной и вычислительной техники (на примере коптера tello).

Методические указания

В ходе выполнения данной работы необходимо решить следующие задачи:

– изучить основные понятия тестирования и расчета надежности программного обеспечения мехатронных и робототехнических систем;

– выбрать одну из моделей надежности и для нее предложить методику тестирования;

– провести тестирование программного обеспечения;

– применить программные методы расчета надежности программного обеспечения.

Многие организации, занимающиеся созданием программного обеспечения, до 50% средств, выделенных на разработку программ, тратят на тестирование, что составляет миллиарды долларов по всему миру в целом. И все же, несмотря на громадные капиталовложения, знаний о сути тестирования явно не хватает, и большинство программных продуктов неприемлемо, ненадежно даже после «основательного тестирования».

Хотя в тестировании можно выделить несколько различных процессов, такие термины, как «тестирование», «отладка», «доказательство», «контроль» и «испытание», часто используются как синонимы и, к сожалению, для разных людей имеют разный смысл. Дадим определения перечисленным терминам, слегка расширив их список.

Тестирование *(testing)* – процесс выполнения программы (или части программы) с намерением (или целью) найти ошибки.

Доказательство *(proof)* – попытка найти ошибки в программе безотносительно к внешней для программы среде. Большинство методов доказательства предполагает формулировку утверждений о поведении программы и затем вывод и доказательство математических теорем о правильности программы. Доказательства могут рассматриваться как форма тестирования, хотя они и не предполагают прямого выполнения программы. Многие исследователи считают доказательство альтернативой тестированию – взгляд во многом ошибочный.

Контроль *(verification)* – попытка найти ошибки, выполняя программу в тестовой или моделируемой среде.

Испытание *(validation)* – попытка найти ошибки, выполняя программу в заданной реальной среде.

Аттестация *(certification)* – авторитетное подтверждение правильности программы. При тестировании с целью аттестации выполняется сравнение с некоторым заранее определенным стандартом.

Отладка *(debugging)* не является разновидностью тестирования. Хотя слова «отладка» и «тестирование» часто используются как синонимы, под ними подразумеваются разные виды деятельности. Тестирование – деятельность, направленная на обнаружение ошибок; отладка направлена на установление точной природы известной ошибки, а затем – на исправление этой ошибки. Эти два вида деятельности связаны – результаты тестирования являются исходными данными для отладки.

Эти определения представляют один взгляд на тестирование – со стороны среды, на которую оно опирается. Другой ряд определений, приведенный ниже, охватывает вторую сторону тестирования: типы ошибок, которые предполагается обнаружить, и стандарты, с которыми сопоставляются тестируемые программы.

Тестирование модуля, или автономное тестирование *(module testing, unit testing)* – контроль отдельного программного модуля, обычно в изолиро-

ванной среде (т.е. изолированно от всех остальных модулей). Тестирование модуля иногда включает также математическое доказательство.

Тестирование сопряжений *(integration testing)* – контроль сопряжений между частями системы (модулями, компонентами, подсистемами).

Тестирование внешних функций *(external function testing)* – контроль внешнего поведения системы, определенного внешними спецификациями.

Комплексное тестирование *(system testing)* – контроль и(или) испытание системы по отношению к исходным целям. Комплексное тестирование является процессом контроля, если оно выполняется в моделируемой среде, и процессом испытания, если выполняется в среде реальной, жизненной.

Тестирование приемлемости *(acceptance testing)* – проверка соответствия программы требованиям пользователя.

Тестирование настройки *(installation testing)* – проверка соответствия каждого конкретного варианта установки системы с целью выявить любые ошибки, возникшие в процессе настройки системы.

Одним из важных аспектов тестирования является последовательность слияния всех модулей в систему или программу. Выбор этой последовательности является одним из самых жизненно важных решений, принимаемых на этапе тестирования, поскольку он определяет форму, в которой записываются тесты, типы необходимых инструментов тестирования, последовательность программирования модулей, а также тщательность и экономичность всего этапа тестирования. По этой причине такое решение должно приниматься на уровне проекта в целом и на достаточно ранней его стадии.

Тестирование модулей (или блоков) представляет собой процесс тестирования отдельных подпрограмм или процедур программы. Здесь подразумевается, что, прежде чем начинать тестирование программы в целом, следует протестировать отдельные небольшие модули, образующие эту программу. Такой подход мотивируется тремя причинами. Во-первых, появляется возможность управлять комбинаторикой тестирования, поскольку первоначально внимание концентрируется на небольших модулях программы. Во-вторых, облегчается задача отладки программы, т.е. обнаружение места ошибки и исправление

текста программы. В-третьих, допускается параллелизм, что позволяет одновременно тестировать несколько модулей.

Цель тестирования модулей – сравнение функций, реализуемых модулем, со спецификациями его функций или интерфейса.

Имеется большой выбор возможных подходов, которые могут быть использованы для слияния модулей в более крупные единицы. В большинстве своем они могут рассматриваться как варианты шести основных подходов, описанных ниже.

Реализация процесса тестирования модулей опирается на два ключевых положения: построение эффективного набора тестов и выбор способа, посредством которого модули комбинируются при построении из них рабочей программы. Рассмотрим два подхода к комбинированию модулей: пошаговое и монолитное тестирование.

Первый подход обычно называют ***монолитным методом***, или ***методом «большого удара»***, при тестировании и сборке программы;

второй подход известен как ***пошаговый метод*** тестирования или сборки.

Метод пошагового тестирования предполагает, что модули тестируются не изолированно друг от друга, а подключаются поочередно для выполнения теста к набору уже ранее оттестированных модулей. Пошаговый процесс продолжается до тех пор, пока к набору оттестированных модулей не будет подключен последний модуль.

Детального разбора обоих методов мы делать не будем, приведем лишь некоторые общие выводы.

1. Монолитное тестирование требует больших затрат труда. При пошаговом же тестировании «снизу вверх» затраты труда сокращаются.
2. Расход машинного времени при монолитном тестировании меньше.
3. Использование монолитного метода предоставляет большие возможности для параллельной организации работы на начальной фазе тестирования (тестирования всех модулей одновременно). Это положение может иметь важное значение при выполнении больших проектов, в которых много модулей и много исполнителей, поскольку численность персонала, участвующего в проекте, максимальна на начальной фазе.

4. При пошаговом тестировании раньше обнаруживаются ошибки в интерфейсах между модулями, поскольку раньше начинается сборка программы. В противоположность этому при монолитном тестировании модули «не видят друг друга» до последней фазы процесса тестирования.

5. Отладка программ при пошаговом тестировании легче. Если есть ошибки в межмодульных интерфейсах, а обычно так и бывает, то при монолитном тестировании они могут быть обнаружены лишь тогда, когда собрана вся программа. В этот момент локализовать ошибку довольно трудно, поскольку она может находиться в любом месте программы. Напротив, при пошаговом тестировании ошибки такого типа в основном связаны с тем модулем, который подключается последним.

6. Результаты пошагового тестирования более совершенны.

Существует две возможные стратегии тестирования: ***нисходящее*** и ***восходящее.***

При восходящем подходе программа собирается и тестируется «снизу вверх». Только модули самого нижнего уровня («терминальные» модули; модули, не вызывающие других модулей) тестируются изолированно, автономно. После того как тестирование этих модулей завершено, вызов их должен быть так же надежен, как вызов встроенной функции языка или оператор присваивания. Затем тестируются модули, непосредственно вызывающие уже проверенные. Эти модули более высокого уровня тестируются не автономно, а вместе с уже проверенными модулями более низкого уровня. Процесс повторяется до тех пор, пока не будет достигнута вершина. Здесь завершается и тестирование модулей, и тестирование сопряжений программы.

При восходящем тестировании для каждого модуля необходим *драйвер:* нужно подавать тесты в соответствии с сопряжением тестируемого модуля! Одно из возможных решений – написать для каждого модуля небольшую ведущую программу. Тестовые данные представляются как «встроенные» непосредственно в эту программу переменные и структуры данных, и она многократно вызывает тестируемый модуль, с каждым вызовом передавая ему новые тестовые данные. Имеется и лучшее решение: воспользоваться программой тестирования модулей – это инструмент тестирования, позволяющий

описывать тесты на специальном языке и избавляющий от необходимости писать драйверы.

Нисходящее тестирование (называемое также нисходящей разработкой) не является полной противоположностью восходящему, но в первом приближении может рассматриваться как таковое. При нисходящем подходе программа собирается и тестируется «сверху вниз». Изолированно тестируется только головной модуль. После того как тестирование этого модуля завершено, с ним соединяются (например, редактором связей) один за другим модули, непосредственно вызываемые им, и тестируется полученная комбинация. Процесс повторяется до тех пор, пока не будут собраны и проверены все модули.

Нисходящий метод имеет как достоинства, так и недостатки, по сравнению с восходящим. Самое значительное достоинство – то, что этот метод совмещает тестирование модуля, тестирование сопряжений и частично тестирование внешних функций. С этим же связано другое его достоинство: когда модули ввода-вывода уже подключены, тесты можно готовить в удобном виде. Нисходящий подход выгоден также в том случае, когда есть сомнения относительно осуществимости программы в целом или когда в проекте программы могут оказаться серьезные дефекты.

Преимуществом нисходящего подхода очень часто считают отсутствие необходимости в драйверах; вместо драйверов вам просто следует написать «заглушки».

Нисходящий метод тестирования имеет, к сожалению, некоторые недостатки. Основным из них является то, что модуль редко тестируется досконально сразу после его подключения.

Вероятно, самый распространенный подход к интеграции модулей – метод «большого скачка». В соответствии с этим методом каждый модуль тестируется автономно. По окончании тестирования модулей они интегрируются в систему все сразу.

Метод «большого скачка», по сравнению с другими подходами, имеет много недостатков и мало достоинств. Заглушки и драйверы необходимы для каждого модуля. Модули не интегрируются до самого последнего момента,

а это означает, что в течение долгого времени серьезные ошибки в сопряжениях могут оставаться необнаруженными.

Если программа мала (как, например, программа загрузчика) и хорошо спроектирована, метод «большого скачка» может оказаться приемлемым. Однако для крупных программ метод «большого скачка» обычно губителен.

Тестирование методом «сандвича» представляет собой компромисс между восходящим и нисходящим подходами. Здесь делается попытка воспользоваться достоинствами обоих методов, избежав их недостатков.

При использовании этого метода одновременно начинают восходящее и нисходящее тестирование, собирая программу как снизу, так и сверху и встречаясь в конце концов где-то в середине. Точка встречи зависит от конкретной тестируемой программы и должна быть заранее определена при изучении ее структуры. Например, если разработчик может представить свою систему в виде уровня прикладных модулей, затем уровня модулей обработки запросов, затем уровня примитивных функций, то он может решить применять нисходящий метод на уровне прикладных модулей (программируя заглушки вместо модулей обработки запросов), а на остальных уровнях применить восходящий метод.

При тестировании методом «сандвича» возникает та же проблема, что и при нисходящем подходе, хотя здесь она стоит не так остро. Проблема эта в том, что невозможно досконально тестировать отдельные модули. Восходящий этап тестирования по методу сандвича решает эту проблему для модулей нижних уровней, но она может по-прежнему оставаться открытой для нижней половины верхней части программы. В модифицированном методе «сандвича» нижние уровни также тестируются строго снизу вверх. А модули верхних уровней сначала тестируются изолированно, а затем собираются нисходящим методом. Таким образом, модифицированный метод «сандвича» также представляет собой компромисс между восходящим и нисходящим подходами.

Результаты тестирования используются для расчета надежности программного средства. Для расчета надежности используется модель надежности.

Термин ***модель надежности программного обеспечения***, как правило, относится к математической модели, построенной для оценки зависимости

надежности программного обеспечения от некоторых определенных параметров. Значения таких параметров либо предполагаются известными, либо могут быть измерены в ходе наблюдений или экспериментального исследования процесса функционирования программного обеспечения. Данный термин может быть использован также применительно к математической зависимости между определенными параметрами, которые хотя и имеют отношение к оценке надежности программного обеспечения, но, тем не менее, не содержат ее характеристик в явном виде. Например, поведение некоторой ветви программы на подмножестве наборов входных данных, с помощью которых эта ветвь контролируется, существенным образом связано с надежностью программы, однако характеристики этого поведения могут быть оценены независимо от оценки самой надежности. Другим таким параметром является частота ошибок, которая позволяет оценить именно качество систем реального времени, функционирующих в непрерывном режиме, и в то же время получать только косвенную информацию относительно надежности программного обеспечения (например, в предположении экспоненциального распределения времени между отказами).

Одним из видов модели надежности программного обеспечения, которая заслуживает особого внимания, является так называемая ***феноменологическая***, или ***эмпирическая***, модель. При разработке моделей такого типа предполагается, что связь между надежностью и другими параметрами является статической. С помощью подобного подхода пытаются количественно оценить те характеристики программного обеспечения, которые свидетельствуют либо о высокой, либо о низкой его надежности. Так, например, параметр *сложность программы* характеризует степень уменьшения уровня ее надежности, поскольку усложнение программы всегда приводит к нежелательным последствиям, в том числе к неизбежным ошибкам программистов при составлении программ и трудности их обнаружения и устранения. Иначе говоря, при разработке феноменологической модели надежности программного обеспечения стремятся иметь дело с такими параметрами, соответствующее изменение (улучшение) значений которых должно приводить к повышению надежности программного обеспечения.

Модели надежности программных средств (МНПС) подразделяются на аналитические и эмпирические. Аналитические модели дают возможность рассчитать количественные показатели надежности, основываясь на данных о поведении программы в процессе тестирования (измеряющие и оценивающие модели). Эмпирические модели базируются на анализе структурных особенностей программ. Они рассматривают зависимость показателей надежности от числа межмодульных связей, количества циклов в модулях, отношения количества прямолинейных участков программы к количеству точек ветвления и т.д. Часто эмпирические модели не дают конечных результатов показателей надежности, однако они включены в классификационную схему, так как развитие этих моделей позволяет выявлять взаимосвязь между сложностью ПС и его надежностью. Эти модели можно использовать на этапе проектирования ПС, когда осуществлена разбивка на модули и известна его структура. Аналитические модели представлены двумя группами: ***динамические модели*** и ***статические.*** В динамических МНПС поведение ПС (появление отказов) рассматривается во времени. В статических моделях появление отказов не связывают со временем, а учитывают только зависимость количества ошибок от числа тестовых прогонов (по области ошибок) или зависимость количества ошибок от характеристики входных данных (по области данных).

Для использования динамических моделей необходимо иметь данные о появлении отказов во времени. Если фиксируются интервалы каждого отказа, то получается непрерывная картина появления отказов во времени (группа динамических моделей с непрерывным временем). Может фиксироваться только число отказов за произвольный интервал времени. В этом случае поведение ПС может быть представлено только в дискретных точках (группа динамических моделей с дискретным временем). Рассмотрим основные аналитические модели надежности ПС.

Аналитическое моделирование надежности ПС включает четыре шага:

1) определение предположений, связанных с процедурой тестирования ПС;

2) разработка или выбор аналитической модели, базирующейся на предположениях о процедуре тестирования;

3) выбор параметров моделей с использованием полученных данных;

4) применение модели – расчет количественных показателей надежности по модели.

ДИНАМИЧЕСКИЕ МОДЕЛИ НАДЕЖНОСТИ

Модель Шумана. Исходные данные для модели Шумана, которая относится к динамическим моделям дискретного времени, собираются в процессе тестирования ПС в течение фиксированных или случайных временных интервалов. Каждый интервал – это стадия, на которой выполняется последовательность тестов и фиксируется некоторое число ошибок.

Модель Шумана может быть использована при определенным образом организованной процедуре тестирования. Использование модели Шумана предполагает, что тестирование проводится в несколько этапов. Каждый этап представляет собой выполнение программы на полном комплексе разработанных тестовых данных. Выявленные ошибки регистрируются (собирается статистика об ошибках), но не исправляются. По завершении этапа на основе собранных данных о поведении ПС на очередном этапе тестирования может быть использована модель Шумана для расчета количественных показателей надежности. После этого исправляются ошибки, обнаруженные на предыдущем этапе, при необходимости корректируются тестовые наборы и проводится новый этап тестирования. При использовании модели Шумана предполагается, что исходное количество ошибок в программе постоянно и в процессе тестирования может уменьшаться по мере того, как ошибки выявляются и исправляются. Новые ошибки при корректировке не вносятся. Скорость (частота) обнаружения ошибок пропорциональна числу оставшихся ошибок. Общее число машинных инструкций в рамках одного этапа тестирования постоянно.

В процессе тестирования собирается информация о времени и количестве ошибок на каждом прогоне, т.е. общее время тестирования τ складывается из времени τ_i, $i=\overline{1,n}$, каждого прогона:

$$\tau=\tau_1+\tau_2+\tau_3+\ldots+\tau_n. \qquad (1)$$

Интенсивность появления ошибок λ_i на i-м прогоне, можно вычислить как число ошибок в единицу времени:

$$\lambda_i = \frac{\sum_{j=1}^{k} A_j}{\tau_i}, \qquad (2)$$

где A_j – j-я ошибка; k – общее число ошибок на i-м прогоне.

Имея данные по двум различным прогонам можно вычислить число ошибок в ПС до начала тестирования $E_т$ [2]

$$E_т = \frac{I_т\left[\lambda_2 / \lambda_1 \varepsilon_c(\tau_1) - \varepsilon_c(\tau_2)\right]}{\left(\lambda_2 / \lambda_1\right) - 1}, \qquad (3)$$

где I_T – общее число машинных команд, которое предполагается постоянным в рамках этапа тестирования, причем данные для двух различных моментов тестирования τ_1 и τ_2 выбираются с учетом требования $\varepsilon_c(\tau_2) > \varepsilon_c(\tau_1)$, где $\varepsilon_c(\tau_2)$, $\varepsilon_c(\tau_1)$ – число ошибок в расчете на команду в машинном языке, обнаруженное в течение времени тестирования τ_2, τ_1 соответственно.

Если время работы ПС без отказа t отсчитывается от точки $t = 0$, а τ остается фиксированным; функция надежности, или вероятность безотказной работы на интервале времени от 0 до t, равна [2]

$$R(t, \tau) = \exp\{C[E_т / I_т - \varepsilon_c(\tau)]t\}, \qquad (4)$$

где

$$C = \frac{\lambda}{\left[E_т / I_т - \varepsilon_c(\tau)\right]}. \qquad (5)$$

Модель La Padula. По этой модели выполнение последовательности тестов производится в m этапов. Каждый этап заканчивается внесением изменений (исправлений) в ПС. Функция надежности базируется на числе ошибок, обнаруженных в ходе каждого тестового прогона.

Надежность ПС $R(t)$ в течение i-го этапа вычисляется по формуле

$$R(t) = R(\infty) - A/i, \quad i = 1, 2\ldots,$$

где A – параметр роста; $R(\infty) = \lim_{i \to \infty} R(i)$ – предельная надежность ПС.

Неизвестные величины A и $R(\infty)$ можно вычислить, решив следующие уравнения [2] (получены эмпирическим путем):

$$\sum_{i=1}^{m}\left\{\frac{S_i - m_i}{S_i} - R(\infty) + \frac{A}{i}\right\} = 0; \tag{6}$$

$$\sum_{i=1}^{m}\left[\left(\frac{S_i - m_i}{S_i} - R(\infty) + \frac{A}{i}\right)\left(\frac{1}{i}\right)\right] = 0, \tag{7}$$

где S_i – число тестов; m_i – число отказов во время i-го этапа; m – число этапов; $i = 1, 2, \ldots, m$.

Определяемый по этой модели показатель есть надежность ПС на i-м этапе: $R(t) = R(\infty) - A/(i)$, $i = m + 1, m + 2 \ldots$.

Преимущество модели заключается в том, что она является прогнозной и, основываясь на данных, полученных в ходе тестирования, дает возможность предсказать вероятность безотказной работы программы на последующих этапах ее выполнения.

Статические модели надежности. Статические модели надежности принципиально отличаются от динамических прежде всего тем, что в них не учитывается время появления ошибок в процессе тестирования. Эти модели строятся на твердом статистическом фундаменте.

Модель Коркорэна. Модель Коркорэна относится к статистическим моделям надежности ПС, так как в ней не используются параметры времени тестирования и учитывается только результат N испытаний, в которых выявлено N_i ошибок i-го типа. Модель использует изменяющиеся вероятности отказов для различных типов ошибок.

По модели Коркорэна оценивается вероятность R безотказного выполнения программы на момент оценки:

$$R = \frac{N_0}{N} + \sum_{i=1}^{k} Y_i(N_i - 1)/N, \tag{8}$$

где N_0 – число безотказных выполнений программы; N – общее число прогонов; k – априори известное число типов ошибок.

$$Y_i = \begin{cases} a_i, & \text{если } N_i > 0; \\ 0, & \text{если } N_i = 0, \end{cases}$$

a_i – вероятность выявления при тестировании ошибки i-го типа.

В этой модели вероятность a_i должна оцениваться на основе априорной информации или данных предшествующего периода функционирования однотипных программных средств.

Модель Нельсона. Данная модель при расчете надежности ПС учитывает вероятность выбора определенного тестового набора для очередного выполнения программы.

Предполагается, что область данных, необходимых для выполнения тестирования программного средства, разделяется на K взаимоисключающих подобластей Z_i, i = 1, 2, …, k. Пусть P_i – вероятность того, что набор данных Z_i будет выбран для очередного выполнения программы. Предполагая, что к моменту оценки надежности было выполнено N_i прогонов программы на Z_i наборе данных и из них n_i количество прогонов закончилось отказом, надежность ПС в этом случае равна

$$R = 1 - \sum_{i=1}^{k} \frac{n_i}{N_i} P_i. \qquad (9)$$

На практике вероятность выбора очередного набора данных для прогона (P_i) определяется путем разбиения всего множества значений входных данных на подмножества и нахождения вероятностей того, что выбранный для очередного прогона набор данных будет принадлежать конкретному подмножеству. Определение этих вероятностей основано на эмпирической оценке вероятности появления тех или иных входов в реальных условиях функционирования.

Рассмотрим пример расчета надежности программного средства для управления коптером в ручном режиме. Для расчета будем использовать модель Коркорэна. Провели 50 прогонов программного средства, которые предполагали взлет коптера, получение видеоизображения с камеры, тестирование кнопок управления (вверх/вниз, влево/вправо, вперед/назад и повороты против/по часовой стрелке). При этом из 50 прогонов 3 раза коптер не взлетел. В 10 прогонах регистрировали отказы программы (невыполнение команды) в журнале (см. таблицу). Появление каждого вида отказа для простоты расчетов будем считать равновероятным.

Таблица

Тип ошибки (невыполнение команды)	Количество отказов N_i	Вероятность отказов
вверх/вниз	2	0,2
влево/вправо	3	
вперед/назад	1	
против/по	1	
нет видеосигнала	1	
Итого	8	

Вычислим, с какой вероятностью наше программное средство будет работать безотказно.

$$\mathrm{R} = \frac{N_0}{N} + \sum_{i=1}^{k} Y_i (N_i - 1) / N,$$

$$R = (50 - 3 - 10)/50 + 0{,}2(1/50 + 2/50) = 0{,}752.$$

Осуществите расчет надежности программного средства, для чего напишите программу для автоматизации расчета в Matlab или MathCad.

ЛАБОРАТОРНЫЕ РАБОТЫ

Лабораторная работа 1

ПРИМЕНЕНИЕ ПРОГРАММНЫХ МЕТОДОВ ПРОЕКТИРОВАНИЯ ПОЛЬЗОВАТЕЛЬСКОГО ИНТЕРФЕЙСА ДЛЯ УПРАВЛЕНИЯ МЕХАТРОННЫМИ И РОБОТОТЕХНИЧЕСКИМИ СИСТЕМАМИ, СРЕДСТВАМИ АВТОМАТИКИ И ИЗМЕРИТЕЛЬНОЙ ТЕХНИКИ

Цель работы – овладеть навыком применения программных методов проектирования пользовательского интерфейса для управления мехатронными и робототехническими системами, средствами автоматики и измерительной техники.

Задачи работы:

1. Изучить методические указания.
2. Выполнить разработку программно-алгоритмического обеспечения в соответствии с утвержденным техническим заданием.

Методические указания

Разработка программно-алгоритмического средства должно выполняться в соответствии с документом «Техническое задание», разработанным в ходе выполнения практической работы 1.

В данных методических указаниях основные моменты разработки приведены на примере системы управления летающей робототехнической системой на базе квадракоптера Tello. Для работы необходима программа QtDesigner любой интерпретатор языка Python и библиотеки *PyQt5, djitellopy, opencv*.

Используемые в работе понятия:

GUI (graphical user interface) – графический интерфейс пользователя, система средств для взаимодействия пользователя с электронными устройствами, основанная на представлении всех доступных пользователю системных

объектов и функций в виде графических компонентов экрана (окон, значков, меню, кнопок, списков и т.п.).

Widget (виджет) – примитив графического интерфейса пользователя, имеющий стандартный внешний вид и выполняющий стандартные действия.

ПОРЯДОК РАЗРАБОТКИ ИНТЕРФЕЙСА СИСТЕМЫ УПРАВЛЕНИЯ КВАДРОКОПТЕРОМ TELLO

Программу QtDesigner можно загрузить с официального сайта: https://build-system.fman.io/qt-designer-download. После установки и запуска вы увидите диалоговое окно, спрашивающее о том, какую форму шаблона вы предпочитаете:

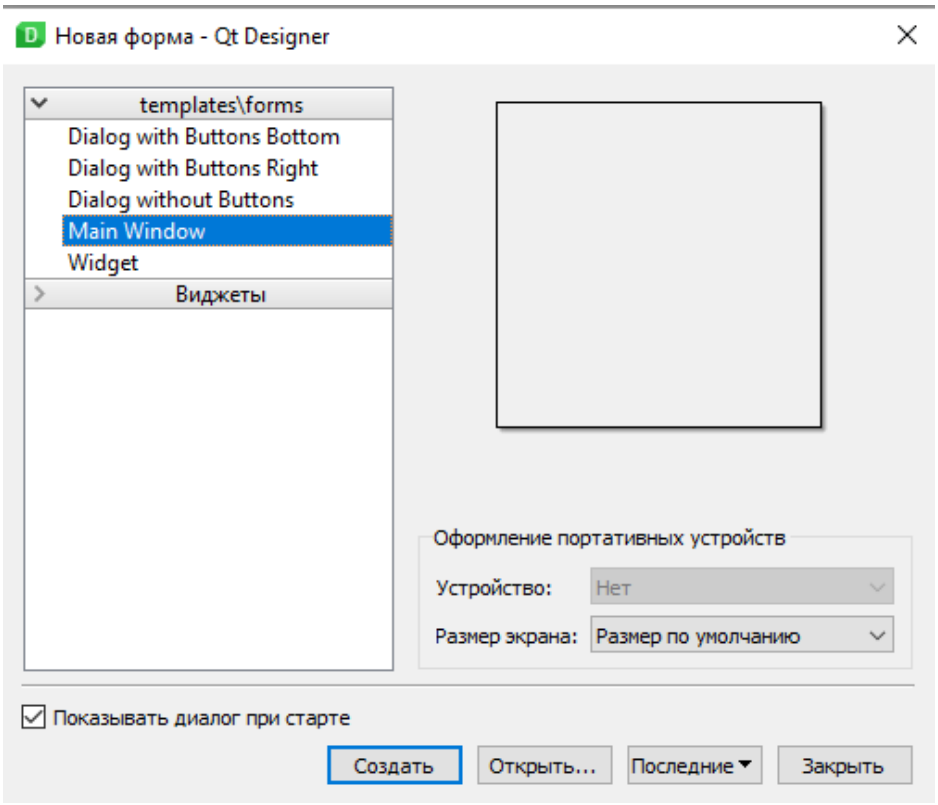


Рис. 1. Формы шаблонов

Существует пять видов доступных шаблонов:

1. Диалоговое окно с кнопками внизу (Dialog with Buttons Bottom): создает форму с кнопками OK и Cancel в правом нижнем углу формы.

2. Диалоговое окно с кнопками справа (Dialog with Buttons Right): создает форму с кнопками OK и Cancel в верхнем правом углу формы.

3. Диалоговое окно без кнопок (Dialog without Buttons): создает пустую форму;

4. Главное окно (Main Window): создает окно с панелью меню и набором инструментов. Унаследовано из QmainWindow.

5. Виджет (Widget): создает виджет, наследуемый из класса Qwidget. Отличается от диалоговых шаблонов тем, что они наследуются из класса QBialog.

Нам необходимо выбрать шаблон *Main Window* и нажать кнопку *create*. Сохраните файл с интерфейсом под именем *copter_control.ui*.

Выбор основных виджетов осуществляется из панели Widget Box (рис. 2).

Свойства виджетов настраиваются с помощью панели Property Editor, расположенной в правой части экрана (рис. 3). В поле *object name* укажите оригинальное имя для каждого виджета, которое далее будет являться именем объекта в программном коде.

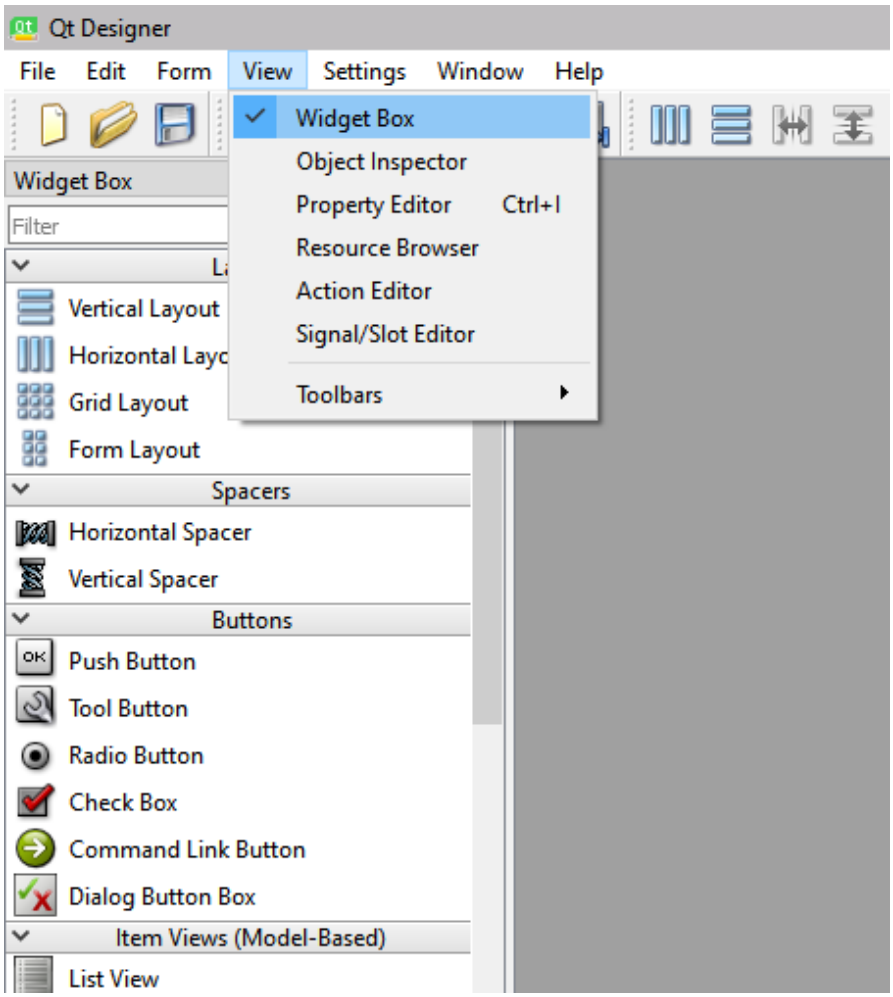


Рис. 2. Панель Widget Box

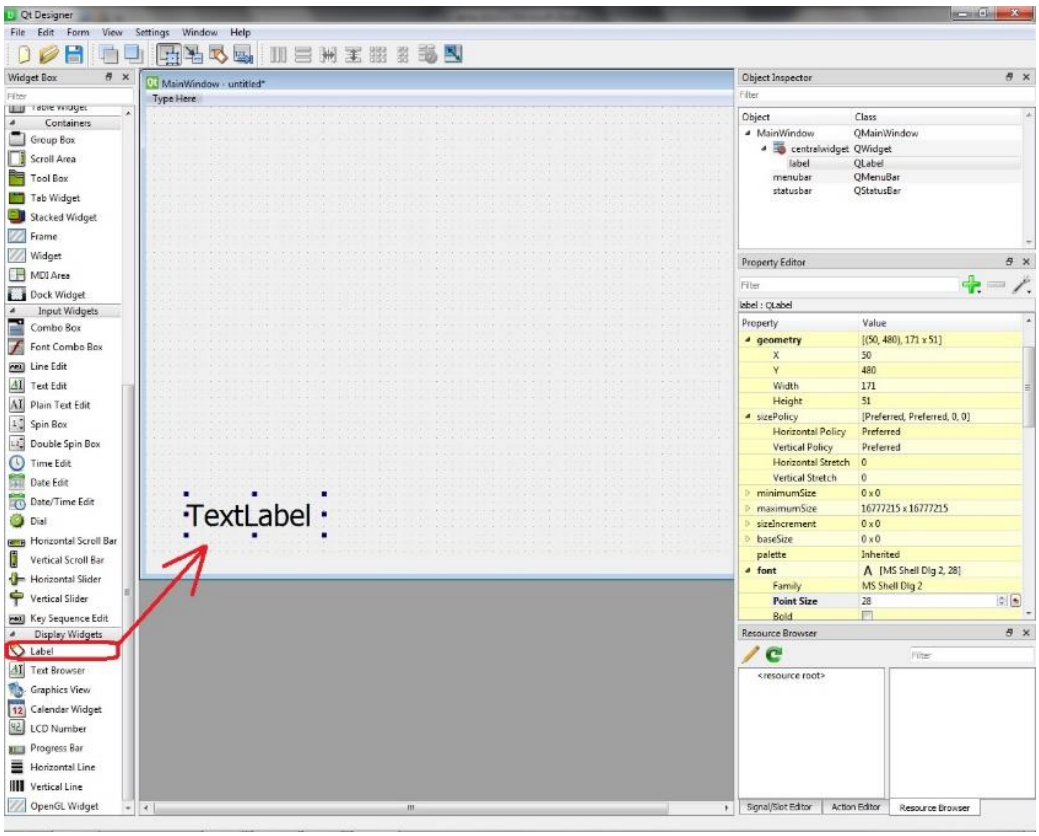


Рис. 3. Создание виджета Label

Далее рассмотрим примеры того, как создавать и настраивать основные виджеты.

1. ***Создание виджета Label.***

Данный элемент используется для отображения текста или рисунка. Перейдите в QtDisigner на панель Widget Box и во вкладке Display Widgets найдите Label, перетащите его в окно си помощью левой кнопки мыши (рис. 3).

Вверху справа располагается ***Object Inspector*** (рис. 4), в нем отображается все что мы поместили в нашу программу:

Рис. 4. Object Inspector

Далее расположен ***Property Editor***, в нем располагаются свойства выбранного витжета. Так, если выбрать наш Label, он будет выглядеть следующим образом (рис. 5).

Рассмотрим некоторые из пунктов:

objectName – определяет имя виджета (то как он будет отображаться в коде программы), по умолчанию стоит label (если добавим второй такой же виджет, он будет называться label_2, и так далее).

В пункте ***geometry*** – задаются размеры виджета.

В пункте ***font*** – задается, каким шрифтом, размером и типом будет выполнена надпись.

Далее прокручиваем вниз, там находится Qlabel.

Пункт ***text*** – определяет, что будет написано в нашем виджете.

Пункт ***alignment*** – определяет выравнивание текста.

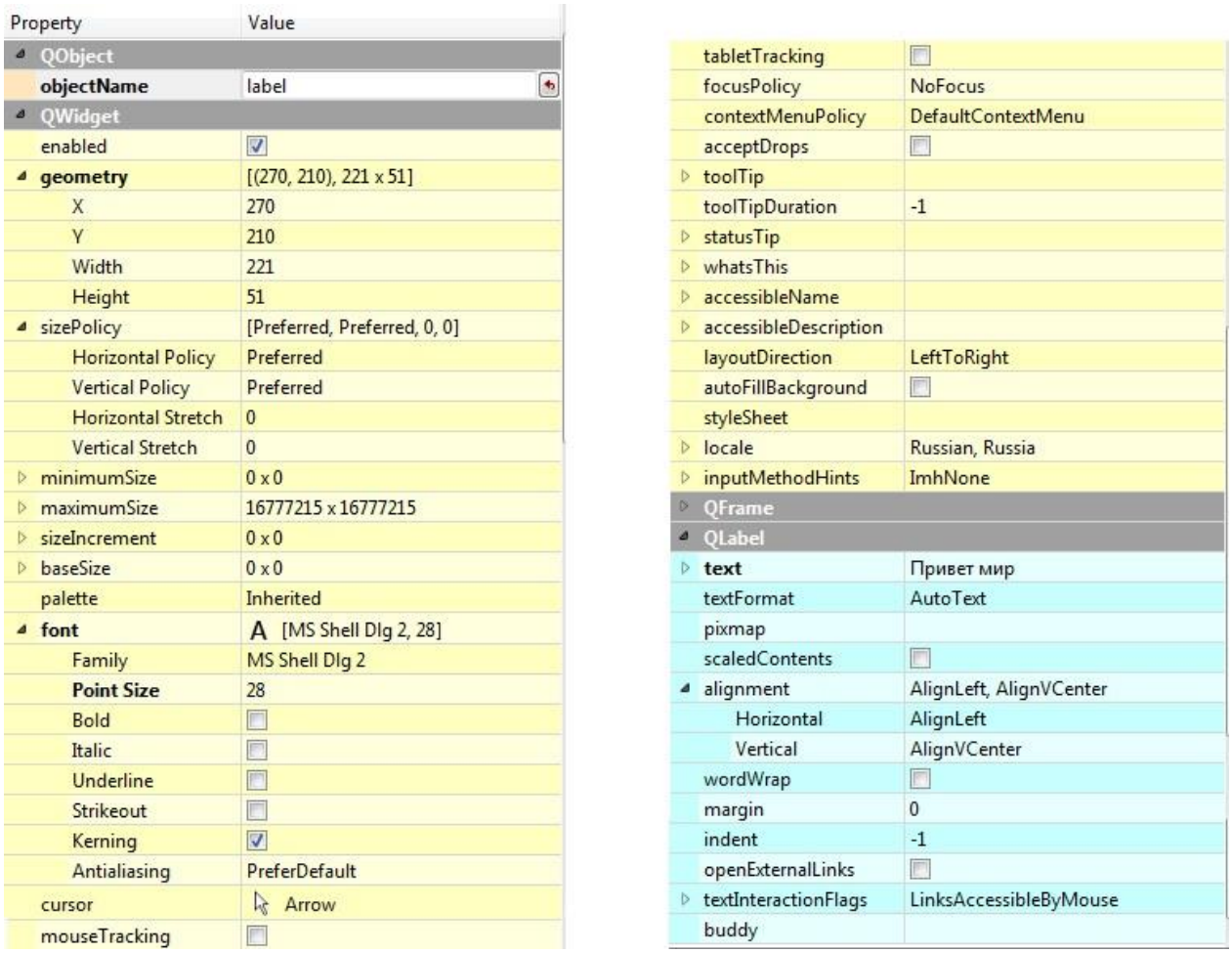


Рис. 5. Property Editor

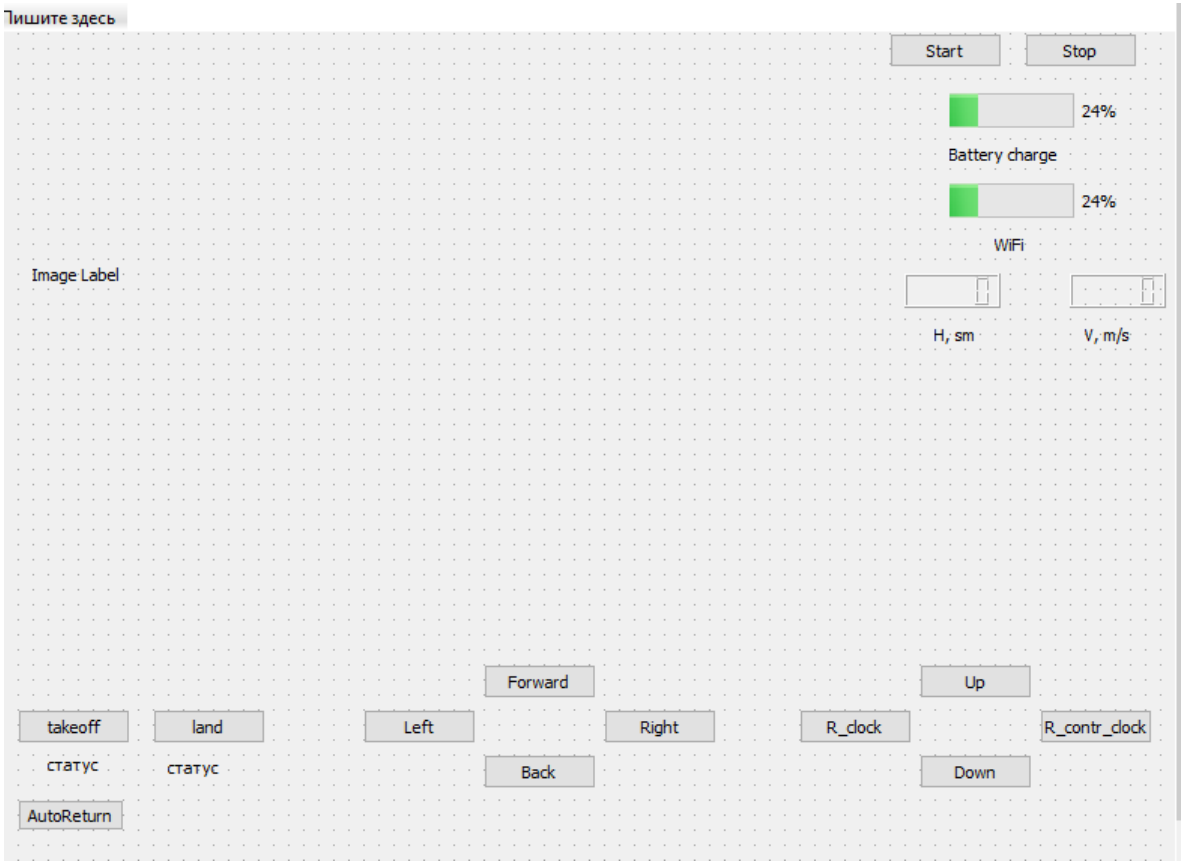


Рис. 6. Примерный вид панели управления

Далее необходимо создать дизайн окна управления квадракоптером в соответствии с требованиями составленного ранее технического задания. В окне должны быть размещены основные органы управления квадракоптером, выводиться изображение с камеры, сигналы с датчиков, предусмотрено включение/отключение функций автопилота и пр.

Примерный вид окна показан на рис. 6.

После окончания разработки внешнего вида интерфейса в QtDisigner сохраните файл copter_control.ui и конвертируйте его в start_panel.py.

Для этого запустите командную строку из папки, где у вас сохранен файл copter_control.ui и выполните команду

```
pyuic5 copter_control.ui -o start_panel.py
```

Далее, создаем папку с проектом, в которую записываем файл start_panel.py. И в этой же папке создаем файл main.py. Далее в файле main.py импортируем виджеты:

```
from PyQt5 import QtWidgets
```

и созданное окно:

```
from start_panel import Ui_MainWindow
import sys
```

Создаем новый класс, который называется `mywindow` путем наследования от класса `QtWidgets.QMainWindow`.

```
class mywindow(QtWidgets.QMainWindow):
 def __init__(self):
super(mywindow, self).__init__()
self.ui = Ui_MainWindow()
self.ui.setupUi(self)
```

Добавляем строки для отображения окна:

```
app = QtWidgets.QApplication([])
application = mywindow()
application.show()
sys.exit(app.exec())
```

Для того, чтобы изменить текст виджета label, необходимо использовать следующую команду:

```
self.ui.label.setText("Взлет")
```

РАЗРАБОТКА ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ ДЛЯ УПРАВЛЕНИЯ КВАДРОКОПТЕРОМ В РУЧНОМ РЕЖИМЕ

Данный этап работы самый продолжительный и трудоемкий. На этом этапе вам необходимо решить несколько задач: подключить необходимые библиотеки, написать код для обработки событий нажатия на кнопки, получить информацию с датчиков и камер и обработать ее. Для успешного выполнения данных задач вначале изучите SDK 2.0 на Tello, находящийся в свободном доступе в Интернет, библиотеку djitellopy.

Подключите библиотеку djitellopy и импортируйте созданный файл с панелью управления.

```
from djitellopy import tello

from start_panel import Ui_MainWindow
```

После этого можно создать объект управления – квадракоптер

```
t = tello.Tello()
t = tello.Tello()
t.connect()
```

и попробовать заставить взлететь его

```
t.takeoff()
```

Ниже приведен полный код, реализующий основные функции управления квадракоптером, – взлет, посадку, полет в различных направлениях (прямо, назад, влево, вправо, развороты), получение кадров с камеры, а также запоминание маршрута полета. Изучите данный код и разработайте собственный, позволяющий реализовать вышеназванные функции. Дополните свой код возможностью автоматического возврата квадракоптера при условии разряда батареи до уровня 30% или при условии, когда уровень сигнала Wi-Fi ниже 40%.

```
from PyQt5 import QtWidgets, QtGui, QtCore
from PyQt5.QtCore import  QTimer
```

```
    from PyQt5.QtGui import QPixmap, QColor, QImage
    from PyQt5.QtWidgets import QWidget, QApplication, QLabel,
QVBoxLayout
    from PyQt5.QtCore import  Qt,QObject
    import cv2

    from djitellopy import tello

    # Импортируем наш шаблон.
    from start_panel import Ui_MainWindow
    import sys,time

    t = tello.Tello()
    tello_map=[]
    #1-вперед, -1 назад; 2,-2 вверх, вниз; 3,-3 влево, вправо;
4,-4 по/против часовой
    commands={1:'t.move_forward(20) ', -1:'t.move_back(20)',
2:'t.move_up(20)', -2:'t.move_down(20)', 3:'t.move_right(20)',-
3:'t.move_left(20)', 4:'tello.rotate_clockwise(20)', -
4:'t.rotate_counter_clockwise(20)'}

    class mywindow(QtWidgets.QMainWindow):
        def __init__(self):
            super(mywindow, self).__init__()
            self.ui = Ui_MainWindow()
            self.ui.setupUi(self)
            self.ui.label.setText("Земля")

            self.refresh=QTimer()

            self.refresh.timeout.connect(self.battery)
            self.ui.progressBar.setValue(100)
            self.cam_refresh=QTimer() #таймера на камеру
            self.cam_refresh.timeout.connect(self.camera)
            self.bat_refresh=QTimer() #таймера на камеру
            self.bat_refresh.timeout.connect(self.charge)

            palette = self.ui.lcdNumber.palette()
            #palette.setColor(palette.WindowText,
QtGui.QColor(85, 85, 255))
            # background color
```

```
            palette.setColor(palette.Background, QtGui.QColor(0,
170, 255))
                # "light" border
            palette.setColor(palette.Light, QtGui.QColor(0, 0, 0))

            self.ui.lcdNumber.setPalette(palette)
            # подключение клик-сигнал к слоту btnClicked
            self.ui.pushButton.clicked.connect(self.btnClicked)

self.ui.pushButton_2.clicked.connect(self.btnClicked_Land)

self.ui.pushButton_3.clicked.connect(self.btnClicked_Start)
            self.ui.Stop.clicked.connect(self.btnClicked_Stop)

self.ui.Forward.clicked.connect(self.btnClicked_Forward)
            self.ui.Back.clicked.connect(self.btnClicked_Back)
            self.ui.Right.clicked.connect(self.btnClicked_Right)
            self.ui.Left.clicked.connect(self.btnClicked_Left)
            self.ui.Up.clicked.connect(self.btnClicked_Up)
            self.ui.Down.clicked.connect(self.btnClicked_Down)

self.ui.Rotate_cl.clicked.connect(self.btnClicked_Rotate_clock)

self.ui.Rotate_contr.clicked.connect(self.btnClicked_Rotate_contr)

self.ui.AutoReturn.clicked.connect(self.btnClicked_AutoReturn)

            self.setWindowTitle("Tello control panel")
        def charge(self):
            ch=t.get_height()
            h=t.query_wifi_signal_noise_ratio()

            self.ui.progressBar.setValue(ch)
            self.ui.progressBar_2.setValue(int(h))
            self.ui.lcdNumber.display(ch)

        def battery(self):

            t.rotate_clockwise(0)
```

```
def camera(self):
    cv_img = t.get_frame_read().frame
    qt_img = self.convert_cv_qt(cv_img)
    self.ui.image_label.setPixmap(qt_img)

def btnClicked_Stop(self):

     t.streamoff()

def btnClicked(self):
    self.ui.label.setText("Взлет")
    t.connect()
    t.takeoff()
    self.ui.label_3.setText("Воздух")
    self.refresh.start(4000)

    # Если не использовать, то часть текста исчезнет.
    self.ui.label.adjustSize()

def btnClicked_Land(self):
    self.ui.label_3.setText("Посадка")
    t.land()
    self.refresh.stop()
    print(tello_map)

def btnClicked_Forward(self):
    t.move_forward(20)
    tello_map.append(1)
def btnClicked_Back(self):
    t.move_back(20)
    tello_map.append(-1)
def btnClicked_Right(self):
    t.move_right(20)
    tello_map.append(3)
def btnClicked_Left(self):
    t.move_left(20)
    tello_map.append(-3)

def btnClicked_Up(self):
    t.move_up(20)
    tello_map.append(2)
def btnClicked_Down(self):
```

```
            t.move_down(20)
            tello_map.append(-2)
        def btnClicked_Rotate_clock(self):
            t.rotate_clockwise(20)
            tello_map.append(4)
        def btnClicked_Rotate_contr(self):
            t.rotate_counter_clockwise(20)
            tello_map.append(-4)

        def btnClicked_Start(self):
            t.connect()

            t.streamon()
            self.cam_refresh.start(40)
            self.bat_refresh.start(6000)

        def convert_cv_qt(self, cv_img):
            """Convert from an opencv image to QPixmap"""
            rgb_image = cv2.cvtColor(cv_img, cv2.COLOR_BGR2RGB)
            h, w, ch = rgb_image.shape
            bytes_per_line = ch * w
            convert_to_Qt_format = QtGui.QImage(rgb_image.data,
w, h, bytes_per_line, QtGui.QImage.Format_RGB888)
            p = convert_to_Qt_format.scaled(500, 281,
Qt.KeepAspectRatio)
            return QPixmap.fromImage(p)

        def btnClicked_AutoReturn(self):
            new_map=reversed(tello_map)
            new_map = [-x for x in new_map]
            print(tello_map)
            print(new_map)

    app = QtWidgets.QApplication([])
    application = mywindow()
    application.show()

    sys.exit(app.exec())
```

Лабораторная работа 3

ПРИМЕНЕНИЕ СИСТЕМ ТЕХНИЧЕСКОГО ЗРЕНИЯ, ОСНОВАННЫХ НА РАЗЛИЧНЫХ ФИЗИЧЕСКИХ ПРИНЦИПАХ ИЗМЕРЕНИЯ, ДЛЯ ПОЛУЧЕНИЯ, ХРАНЕНИЯ И ОБРАБОТКИ ИНФОРМАЦИИ В МЕХАТРОННЫХ И РОБОТОТЕХНИЧЕСКИХ СИСТЕМАХ

Цель работы – научиться применять системы технического зрения для получения, хранения и обработки информации в роботизированном дроне.

Задачи работы:

1. Разработка программного обеспечения для захвата и обработки изображения с камеры квадракоптера.
2. Разработка программного обеспечения для получения информации с ИК-датчика расстояния квадракоптера.
3. Разработка программного обеспечения для получения уровня мощности сигнала Wi-Fi и его обработки.

Методические указания

Вначале рекомендуется ознакомиться с SDK 2.0 Tello и документацией на библиотеку djitellopy.

Захват и обработка изображения с камеры квадракоптера

Для решения первой задачи необходимо получить видеопоток с камеры Tello и вывести кадры в окно. Рассмотрим решение этой задачи. Создадим виджет image_label класса QLabel. Захват изображения с камеры будет осуществляться после нажатия кнопки Start. Добавим эту кнопку и назовем ее pushButton_3. Привяжем функцию, которая будет вызываться при нажатии этой кнопки.

```
self.ui.pushButton_3.clicked.connect(self.btnClicked_Start)
```

Сама функция будет инициировать видеопоток t.streamon() и запускать таймер cam_refresh, срабатывающий с частотой 40 мс. Таймер принадлежит

классу QTimer и определяется как self.cam_refresh=QTimer() при инициализации в методе __init__(self).

```
def btnClicked_Start(self):
    t.connect()
    t.streamon()
    self.cam_refresh.start(40)
```

Срабатывание таймера обеспечивает вызов метода

camera self.cam_refresh.timeout.connect(self.camera)

Метод camera обеспечивает чтение кадров с камеры, конвертацию изображения в формат QPixmap, пригодный для отображения в виджете image_label:

```
def camera(self):
    cv_img = t.get_frame_read().frame
    qt_img = self.convert_cv_qt(cv_img)
    self.ui.image_label.setPixmap(qt_img)
```

Конвертация в формат QPixmap осуществляется в методе convert_cv_qt

```
def convert_cv_qt(self, cv_img):

    rgb_image = cv2.cvtColor(cv_img, cv2.COLOR_BGR2RGB)
    h, w, ch = rgb_image.shape
    bytes_per_line = ch * w
    convert_to_Qt_format = QtGui.QImage(rgb_image.data, w, h,
bytes_per_line, QtGui.QImage.Format_RGB888)
    p = convert_to_Qt_format.scaled(500, 281,
Qt.KeepAspectRatio)
    return QPixmap.fromImage(p)
```

После чего конвертированное изображение помещается в виджет.

image_label self.ui.image_label.setPixmap(qt_img)

Лабораторная работа 4

ПРИМЕНЕНИЕ СТАНДАРТОВ ЕСПД НА РАЗРАБОТКУ ПРОГРАММНО-АЛГОРИТМИЧЕСКОГО ОБЕСПЕЧЕНИЯ МЕХАТРОННЫХ И РОБОТОТЕХНИЧЕСКИХ СИСТЕМ

Цель работы – изучить требования стандартов ЕСПД на разработку алгоритмов, оформление текста программ и описание программ, научиться применять их на практике

Задачи работы:

1. Изучить методические указания.

2. Разработать документы «Описание программы», «Текст программы», «Алгоритм программы» в соответствии со стандартами ЕСПД.

Методические указания

Примерное время выполнения работы составляет 8 ч, включая 4 ч аудиторной и 4 ч самостоятельной, внеаудиторной работы. Вначале рекомендуется ознакомиться с методическими указаниями, изучить нормативную документацию на разработку программных средств.

При разработке документа «Описание программы» следует использовать стандарт ГОСТ 19.402–78 ЕСПД, который определяет состав и требования к содержанию программного документа «Описание программы». Описание программы включает:

1. Общие сведения.
2. Функциональное назначение.
3. Описание логической структуры.
4. Используемые технические средства.
5. Вызов и загрузка.
6. Входные данные.
7. Выходные данные.

В разделе ***Общие сведения*** указывают:

- обозначение и наименование программы;

- программное обеспечение, необходимое для функционирования программы;
- языки программирования, на которых написана программа.

Раздел ***Функциональное назначение*** должен отражать классы решаемых задач и(или) назначение программы, сведения о функциональных ограничениях на применение.

При описании ***логической структуры*** должны быть отражены:

- алгоритм программы;
- используемые методы;
- структура программы с описанием функций составных частей и связей между ними;
- связи программы с другими программами.

В разделе ***Используемые технические средства*** указывают типы ЭВМ и устройств, которые используются при работе программы.

При описании раздела ***Вызов и загрузка*** указывают способ вызова программы с соответствующего носителя данных и входные точки в программу.

Раздел ***Входные данные*** отражает:

– характер, организацию и предварительную подготовку входных данных;

– формат, описание и способ кодирования входных данных.

Раздел ***Выходные данные*** отражает:

– характер и организацию выходных данных;

– формат, описание и способ кодирования выходных данных.

При разработке документа «Текст программы» следует руководствоваться ГОСТ 19.401–78. При разработке алгоритма следует руководствоваться ГОСТ 19.701–90.

КОНТРОЛЬНЫЕ ВОПРОСЫ

1. В соответствии с каким нормативным документом разрабатывается «Описание программы»?
2. Какие разделы входят в документ «Описание программы»?
3. Что такое функциональное назначение программного средства?
4. Какие технические средства используются для работы программных продуктов?
5. Перечислите виды программных документов, разрабатываемых в процессе создания программных средств.
6. Какие разделы содержит программный продукт «Руководство программиста» по ГОСТ 19.504–79 ЕСПД?
7. Какие разделы содержит программный продукт «Руководство оператора» по ГОСТ 19.505–79 ЕСПД?
8. Какие разделы содержит программный продукт «Описание языка» по ГОСТ 19.506–79 ЕСПД?
9. Перечислите стадии разработки программ и программной документации.
10. Перечислите основные этапы работ и их содержание на этапе «Техническое задание».
11. Перечислите основные этапы работ и их содержание на этапе «Эскизный проект».
12. Перечислите основные этапы работ и их содержание на этапе «Технический проект».
13. Перечислите основные этапы работ и их содержание на этапе «Рабочий проект».
14. Перечислите основные этапы работ и их содержание на этапе «внедрение».

МЕТОДИЧЕСКИЕ РЕКОМЕНДАЦИИ ПО ВЫПОЛНЕНИЮ КУРСОВОЙ РАБОТЫ

Целями курсовой работы являются: овладение навыком организации работ по разработке алгоритмов и программ для управления летающими робототехническими системами, либо средствами автоматики на базе логических контроллеров, либо средствами измерительной техники на основе оптических методов контроля.

Тема курсовой работы согласуется обучающимся с преподавателем.

Примерное название темы курсовой работы: **«Программно-алгоритмическое обеспечение для управления *здесь указывается объект управления*»**. Например, **«Программно-алгоритмическое обеспечение для управления *квадракоптером tello в ручном режиме*»** или **«Программный модуль автоматического взлета/посадки *квадракоптера tello*»**.

Требования к основным разделам курсовой работы. Курсовая работа должна содержать 4 главы.

1. В первой главе должен быть проведен обзор современной литературы в предметной области работы (например, если в работе в качестве источника информации используется тепловизионная камера, то должен быть проведен обзор методов теплового контроля и средств для его осуществления).

2. Во второй главе должно быть разработано техническое задание на разработку. Требования к техническому заданию см. в практической работе 1.

3. В третьей главе должна быть разработана программа и программная документация – текст программы и описание программы в соответствии с требованиями стандартов ЕСПД.

4. В четвертой главе должно быть проведено испытание программы, приведены результаты испытаний в виде таблиц, графиков или ином виде, достаточном для доказательства достижения цели работы.

Курсовая работа должна соответствовать выбранной теме, содержать все основные разделы и графический материал в соответствии с заданием,

должна быть оформлена в соответствии с СТО ФГБОУ ВО «ТГТУ» 07–2017 «Выпускные квалификационные работы и курсовые проекты (работы). Общие требования».

ОБЩИЕ РЕКОМЕНДАЦИИ ПО ОРГАНИЗАЦИИ РАБОТ ПО РАЗРАБОТКЕ АЛГОРИТМОВ И ПРОГРАММ В РАМКАХ КУРСОВОЙ РАБОТЫ

1. Определите объект управления. Это может быть коптер, мобильный робот, манипулятор, мехатронное устройство для измерения и контроля качества веществ, материалов или изделий. При этом в качестве объекта управления может быть и виртуальный объект, разработанный, например, в среде Vrep.
2. Определите, какие задачи должен выполнять выбранный вами объект управления. Например, коптер должен иметь функцию автовозврата при потере сигнала Wi-Fi, автоматической посадки на заданную площадку, регистрации объектов на изображении с камеры и т.д.
3. Если задач несколько, то для их решения сформируйте команду разработчиков, распределите задачи между ними. Каждый разработчик может отвечать за разработку отдельного модуля, выполняющего задачу. Договоритесь об общем интерфейсе между модулями.
4. Сформируйте техническое задание на разработку, поставьте сроки выполнения.
5. Не забудьте оставить время на тестирование вашего программного средства и оформление курсовой работы.

ЗАКЛЮЧЕНИЕ

Уважаемый читатель! Надеемся, что вместе мы прошли трудный, но интересный путь от основ создания приложений для управления дроном tello до разработки отдельных документов на программный продукт в соответствии с требованиями ЕСПД и расчета надежности созданных программных средств. Если вы успешно справились с практическими задачами пособия, то готовы к самостоятельному развитию полученных знаний, умений, навыков. Желаем вам творческих успехов в развитии робототехнических и мехатронных систем.

СПИСОК ЛИТЕРАТУРЫ

1. **Балабанов, П. В.** Система автоматизированных расчетов MATLAB : лабораторный практикум / П. В. Балабанов, Н. А. Конышева, Д. А. Любимова. – Тамбов : Изд-во ФГБОУ ВПО «ТГТУ», 2015. – 80 с. – URL : https://clck.ru/36XZ6f (дата обращения: 11.11.2023).

2. **Дивин, А. Г.** Информационно-сенсорные системы мехатроники и роботизированные комплексы в неразрушающем контроле качества : учебное пособие : в 2-х ч. Ч. 1 / А. Г. Дивин, П. В. Балабанов, Д. А. Любимова. – Тамбов : Издательский центр ФГБОУ ВО «ТГТУ», 2022. – 80 с. – URL : https://www.tstu.ru/book/ elib1/exe/2022/Divin.exe (дата обращения: 11.11.2023).

3. **Балабанов, П В.** Техническое зрение робототехнических комплексов: учебное пособие / П. В. Балабанов, А. Г. Дивин, А. С. Егоров. – Тамбов : Издательский центр ФГБОУ ВО «ТГТУ», 2019. – 81 с. – URL : https://tstu.ru/r.php?r=obuch. book.elib1&id=15&year=2019 (дата обращения: 11.11.2023).

4. **Никитина, Т. П.** Программирование. Основы Python для инженеров / Т. П. Никитина, Л. В. Королев // Лань: электронно-библиотечная система. – СПб. : Лань, 2023. – 156 с. – URL : https://e.lanbook.com/book/302720 (дата обращения: 11.11.2023).

5. **ГОСТ 19.106–78.** Единая система программной документации. Требования к программным документам, выполненным печатным способом.

6. **ГОСТ 19.104 –78.** Единая система программной документации. Основные надписи.

7. **API** reference of the DJITelloPy Library [Электронный ресурс]. – URL : https://djitellopy.readthedocs.io/en/latest/ (дата обращения: 11.11.2023).

8. **Уилкс, М.** Профессиональная разработка на Python / М. Уилкс ; пер. с англ. А. А. Слинкина // Лань: электронно-библиотечная система. – М. : ДМК Пресс, 2021. – 502 с. – URL : https://e.lanbook.com/book/241121 (дата обращения: 11.11.2023).

9. **Дьяконов, В. П.** MATLAB 7.*/R2006/R2007: Самоучитель / В. П. Дьяконов // Лань: электронно-библиотечная система. – М. : ДМК Пресс, 2009. – 768 с. – URL : https:// e.lanbook.com/book/1178 (дата обращения: 11.11.2023).

10. **Ян, Э. С.** Программирование компьютерного зрения на языке Python / Э. С. Ян ; пер. с англ. А. А. Слинкина // Лань: электронно-библиотечная система. – М. : ДМК Пресс, 2016. – 312 с. – URL : https://e.lanbook.com/book/93569 (дата обращения: 11.11.2023).

СОДЕРЖАНИЕ

Учебное электронное издание

БАЛАБАНОВ Павел Владимирович
ДИВИН Александр Георгиевич
ЛЮБИМОВА Дарья Александровна

ПРОГРАММИРОВАНИЕ БЕСПИЛОТНОГО ЛЕТАТЕЛЬНОГО АППАРАТА МУЛЬТИРОТОРНОГО ТИПА

Учебное пособие

Редактор Л. В. Комбарова
Графический и мультимедийный дизайнер Т. Ю. Зотова
Обложка, упаковка, тиражирование Л. В. Комбаровой

ISBN 978-5-8265-2689-7

Подписано к использованию 30.11.2023.
Тираж 50 шт. Заказ № 167

Издательский центр ФГБОУ ВО «ТГТУ»
392000, г. Тамбов, ул. Советская, д. 106, к. 14
Телефон 8(4752) 63-81-08
E-mail: izdatelstvo@tstu.ru