

Краткое описание проекта

Данный проект представляет собой комплексную систему голосового управления автономным квадрокоптером, состоящую из двух взаимосвязанных компонентов: Python-приложения для обработки голосовых команд и Arduino-прошивки для управления полетной механикой. Система позволяет пользователю обучать дрон реагировать на персонализированные голосовые команды, преобразовывая их в конкретные последовательности движений. Ключевой особенностью проекта является возможность создания готовых файлов миссий в формате XML, совместимых с популярным программным обеспечением Mission Planner, что позволяет интегрировать голосовое управление в существующие экосистемы автономных полетов. Проект реализует многоуровневую архитектуру, включающую этапы обучения голоса, распознавания команд, обработки естественного языка и преобразования команд в низкоуровневые управляющие сигналы для двигателей дрона.

Система разработана с акцентом на безопасность и надежность, предусматривая механизмы подтверждения команд, защиту от ложных срабатываний и возможность экстренной остановки. Пользовательский интерфейс реализован в виде интерактивного консольного меню, предоставляющего полный контроль над процессом обучения и управления. Особенностью реализации является использование демонстрационного режима работы, позволяющего тестировать систему без необходимости наличия физического дрона и микрофона, что упрощает разработку и отладку. Проект представляет собой полноценный фреймворк для исследований в области человеко-машинного взаимодействия и автономных систем управления.

Логика Python-кода

Python-компонент системы построен по модульному принципу и состоит из трех основных классов: VoiceTrainer, ArduinoCommander и VoiceDroneController. Класс VoiceTrainer отвечает за обработку голосовых данных и реализует механизм обучения распознаванию команд. Его работа начинается с записи аудиосэмплов от пользователя, для каждого из которых извлекаются мел-кепстральные коэффициенты (MFCC) — стандартные в речевой обработке признаки, описывающие спектральные характеристики звука. Эти признаки агрегируются в уникальный "голосовой отпечаток" команды, который сохраняется в сериализованной базе данных вместе с метаданными. Система использует вероятностную модель для сравнения входящего

аудиосигнала с обученными образцами, вычисляя косинусное сходство между векторами признаков.

Класс `ArduinoCommander` инкапсулирует логику взаимодействия с аппаратной частью через последовательный порт. Он реализует протокол обмена текстовыми командами, где каждая голосовая команда маппируется на predetermined набор инструкций для полетного контроллера. Для обеспечения надежности связи класс включает механизмы обработки таймаутов, валидации входящих команд и повторных попыток отправки. В демонстрационном режиме класс имитирует реальное взаимодействие, эмулируя поведение Arduino-устройства, что позволяет тестировать логику управления без физического подключения.

Ядром системы является класс `VoiceDroneController`, который координирует работу всех компонентов. Он управляет процессом привязки голосовых команд к конкретным действиям дрона, используя словарь стандартных соответствий (например, "взлет" → "TAKEOFF"). При получении голосовой команды контроллер инициирует каскад обработки: сначала происходит распознавание речи через `VoiceTrainer`, затем поиск соответствующего действия в таблице маппинга, и наконец — отправка команды через `ArduinoCommander`. Особого внимания заслуживает модуль генерации XML-миссий, который преобразует последовательности движений в структурированные файлы, содержащие waypoints с координатами, высотами и MAVLink-командами в формате, понятном программному обеспечению `ArduPilot`.

Архитектура Python-приложения реализует шаблон "Фасад", скрывая сложность внутренних преобразований за простым интерфейсом главного меню. Система поддерживает два режима работы: интерактивное обучение новых команд через пошаговый диалог с пользователем и автоматическое выполнение предварительно обученных последовательностей. Особенностью реализации является обработка исключительных ситуаций на всех уровнях — от ошибок записи аудио до проблем с серийной связью, что обеспечивает устойчивость системы к сбоям.

Логика Arduino-кода

Arduino-компонент представляет собой низкоуровневую систему реального времени, написанную на C++ и развертываемую на микроконтроллере, непосредственно управляющем квадрокоптером. Центральным элементом кода является алгоритм PID-регулирования, который непрерывно обрабатывает данные с инерциального измерительного модуля (MPU6050) для стабилизации положения дрона в пространстве. Код реализует комплементарный фильтр, объединяющий показания гироскопа и акселерометра: гироскоп предоставляет точные данные о

скорости вращения, но подвержен дрейфу, в то время как акселерометр дает стабильные, но шумные данные об ориентации относительно силы тяжести. Фильтр настраивается коэффициентом 0.98 для гироскопа и 0.02 для акселерометра, достигая оптимального баланса между точностью и стабильностью.

Система управления построена по двухрежимной архитектуре. В ручном режиме значения управляющих сигналов считываются с PWM-приемника через аппаратные прерывания, что обеспечивает минимальную задержку реакции. Канал прерывания фиксирует фронты импульсов и вычисляет их длительность, преобразуя ее в значения в диапазоне 1000-2000 микросекунд, соответствующие стандарту RC-управления. В голосовом режиме команды принимаются через SoftwareSerial-интерфейс от Raspberry Pi, парсятся в текстовом формате и транслируются в целевые значения для PID-контроллеров. Реализован конечный автомат с состояниями TAKEOFF, LAND, HOVER и другими, каждый из которых определяет специфическое поведение системы управления.

Моторный миксинг реализован через матричное преобразование, где значения тяги, крена, тангажа и рыскания распределяются между четырьмя двигателями согласно конфигурации квадрокоптера типа "X". Формулы расчета учитывают расположение и направление вращения пропеллеров: передние моторы получают отрицательные поправки по тангажу для наклона вперед, левые моторы — положительные по крену для наклона вправо, и т.д. Все вычисления выполняются в целочисленной арифметике для оптимизации производительности, с последующим ограничением результатов в диапазоне 1100-2000 микросекунд, безопасном для электронных регуляторов скорости.

Система безопасности включает несколько уровней защиты. Watchdog-таймер отслеживает зависания программного обеспечения, механизм failsafe активируется при потере сигнала от наземной станции, а экстренная остановка инициируется при определенной комбинации положений стиков передатчика. Код оптимизирован для работы в жестких временных рамках — основной цикл управления выполняется с частотой 250 Гц (период 4000 микросекунд), что обеспечивает достаточную скорость реакции для стабильного полета. Связь с верхним уровнем осуществляется через асинхронный обмен сообщениями, где каждая команда подтверждается обратной связью о текущем состоянии системы.

Интеграция и взаимодействие компонентов

Взаимодействие между Python-приложением и Arduino-прошивкой организовано через асинхронный последовательный протокол на скорости 9600 бод. Протокол обмена использует текстовый формат с разделителем новой строки, где каждая команда представляет собой ключевое слово из predetermined набора. Система реализует модель "запрос-ответ" с таймаутом: Raspberry Pi отправляет команду и ожидает подтверждения в течение 3 секунд, после чего автоматически возвращается в ручной режим управления. Такая архитектура обеспечивает отказоустойчивость — при потере связи дрон либо завершает текущую операцию и переходит в режим зависания, либо инициирует процедуру безопасной посадки.

Процесс обучения голосовых команд организован как итеративный диалог с пользователем. Система запрашивает произнесение команды несколько раз с вариациями интонации, что позволяет построить робастную модель распознавания, устойчивую к изменениям громкости, тембра и фоновому шуму. Собранные аудиоданные проходят предобработку: нормализацию амплитуды, фильтрацию низких частот для устранения постоянной составляющей и сегментацию на фреймы для анализа. Извлеченные признаки сохраняются в структурированном формате вместе с метаданными для возможного переобучения или анализа.

Генерация XML-миссий представляет собой трансляцию высокоуровневых команд в низкоуровневые инструкции автопилота. Каждое движение (например, "вперед на 2 метра") конвертируется в waypoint с относительными координатами и параметрами MAVLink-команд. Система автоматически добавляет точки взлета и посадки, формируя завершенную миссию. Созданные файлы используют стандартную структуру Mission Planner, что позволяет загружать их напрямую в программу планирования полетов или на бортовой компьютер дрона через MAVLink-протокол.

Архитектура проекта демонстрирует принципы распределенных вычислений, где задачи разделены между устройствами согласно их вычислительным возможностям: Raspberry Pi выполняет ресурсоемкие операции распознавания речи и планирования, в то время как Arduino отвечает за критичные ко времени задачи управления двигателями. Такое разделение обеспечивает как высокую производительность системы обработки голоса, так и гарантированное время отклика контуров управления, что является ключевым требованием для устойчивого полета мультироторных аппаратов.

Подводные камни

Проект сдаётся как концепция, а не решение под ключ – не успели сориентироваться для решения всех проблем и задач. В добавок навыки программирования оставляют желать лучшего. Основной прокол – отсутствие железа. Пока запчасти добирались времени на тесты почти не осталось. Основная проблема – где, как и куда размещать логику – микрофон не разместишь на квадракоптер, значит он привязан к своему пульта (или зарядной станции, где установлен микрофон). Очень тяжело давалось обучение по штаммам голоса. И много чего ещё, но работай над интересной задачей остались довольны с коллегой.