УДК 004.624, 004.45

ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ ЁМКОСТНОЙ СЕНСОРНОЙ СИСТЕМЫ

С. Р. КИСЕЛЁВ
(Представлено: Н. В. ВАБИЩЕВИЧ)

Представлено встраиваемое программное обеспечение, разработанное на базе платформы Arduino Uno, для реализации логики управления ёмкостными сенсорами, алгоритма автоматической калибровки и протокола беспроводной связи. Разработанное на Python кроссплатформенное приложение позволяет осуществлять мониторинг работы панели в реальном времени. Представленная разработка может быть интегрирована в лабораторное оборудование и образовательные проекты, что позволит облегчить обучающимся освоение технологий сенсорного управления.

Введение. С начала 2000-х в мире наблюдается активный рост применения технологий бесконтактного управления электронными устройствами, что обусловлено такими их преимуществами по сравнению с кнопочным управлением как надёжность, долговечность, соблюдение стерильности, минимизация физического износа устройств [1]. Особый интерес проявляется в настоящее время к поиску и разработке компактных и недорогих устройств сенсорного управления. Автором настоящей работы был создан прототип ёмкостной панели управления на базе платформы Arduino Uno, включающий в себя несколько сенсорных зон и систему обработки жестов, в связи с чем появилась задача написания программного обеспечения для обеспечения работы сенсоров и визуализации данных, поступающих с сенсорной панели, диагностики системы и анализа её работы в реальном времени.

Цель работы состояла в разработке встраиваемого программного обеспечения (ПО) для управления ёмкостными сенсорами прототипа беспроводной ёмкостной панели управления, разработанной на базе Arduino Uno.

Основная часть. *Встраиваемое ПО передатчика.* Программа, являющаяся основой ПО сенсорной панели, состоит из двух основных процедур: *setup()*, выполняемой однократно при запуске, и *loop()*, работающей в бесконечном цикле. Код написан на языке C++ [2] в среде Arduino IDE [3] с использованием ключевых библиотек: *CapacitiveSensor.h* для работы с ёмкостными датчиками, *RF24.h* для управления трансивером nRF24L01+ и *SPI.h* для реализации низкоуровневого интерфейса связи с радиомодулем [4, 5].

Программно реализовано применение конечного автомата с двумя состояниями: калибровка и рабочий режим.

Алгоритм калибровки активируется при старте системы и выполняет следующие действия:

– Инициализация периферии - настройка цифровых портов, инициализация аппаратного интерфейса SPI для обмена данными с радиомодулем, настройка самого радиомодуля.

– Отключение автокалибровки библиотеки для предотвращения конфликта алгоритмов, так как встроенный механизм постоянно адаптирует базовый уровень под средние значения, в то время как наша реализация использует пиковые значения фона для установки статичного порога срабатывания.

– Определение базового уровня: В течение 100 итераций система считывает показания с каждого сенсора с 10-кратным усреднением (*capacitiveSensor(10)*), определяя максимальные зафиксированные значения фона.

– Установка порога с гистерезисом: К полученным максимальным значениям добавляется статичное значение гистерезиса (40 единиц). Этот приём исключает ложные срабатывания от незначительных колебаний окружающей среды и обеспечивает чёткий момент активации.

Рабочий режим реализован в функции *loop()* и представляет собой следующий алгоритм:

– Опрос сенсоров - циклическое чтение текущих значений с всех трёх сенсоров с 10-кратным усреднением.

– Формирование и отправка данных - после считывания сигналов формируется сообщение, которое немедленно передаётся по радиоканалу методом *radio.write()*. Отправка каждого пакета данных осуществляется через низкоуровневый интерфейс SPI, который обеспечивает высокоскоростную и надёжную передачу.

– Введение задержки для стабилизации измерений (исключение влияние заряда, накопленного на сенсоре, на следующее измерение) и снижения общего энергопотребления системы за счёт уменьшения времени активности процессора.

Фундаментальным фактором, обеспечивающим работу всей системы, является корректная инициализация объектов сенсоров с указанием физических выводов микроконтроллера. Ключевым программным компонентом, обеспечивающим работу ёмкостных сенсоров в настоящем проекте является библиотека *CapacitiveSensor*, преимуществом использования которой стало обеспечение точного измерения

малых значений электрической ёмкости. Использование указанной библиотеки позволило создать простой и эффективный алгоритм управления сенсорными элементами разработанной системы.

Прежде всего, для обеспечения независимости управления каждым отдельным сенсором были созданы программные объекты, каждый из которых связан с конкретной парой физических выводов микроконтроллера, используемых в цифровом режиме.

Принцип работы библиотеки *CapacitiveSensor* основан на методе заряда и разряда RC-цепи. Алгоритм измеряет время, необходимое для разряда ёмкостного электрода через резистор известного номинала. Основной метод измерения *capacitiveSensor (10)* выполняет 10 последовательных циклов измерений с последующим усреднением результата, что позволяет повысить стабильность показаний за счёт подавления случайных выбросов и уменьшить влияние высокочастотных помех и получить репрезентативное значение ёмкости.

Библиотека обеспечивает разрешающую способность порядка 0.1 пикофарада, что достаточно для уверенного детектирования приближения руки на расстоянии ≈4 см от поверхности используемых сенсоров. Стабильность показаний достигается благодаря как программному усреднению, так и правильной организации заземления экранирующих элементов.

Библиотека *CapacitiveSensor* возвращает результат измерения в условных единицах (количество тактов, затраченных на процесс заряда и разряда RC-цепочки). В основе расчёта лежит подсчёт числа циклов, которое требуется для разряда сенсорного электрода через резистор до низкого логического уровня. Таким образом, получаемые значения являются относительными и зависят от параметров RC-цепи, но прямо коррелируют с изменением ёмкости сенсора. В практическом применении это выражается в том, что рост значения соответствует увеличению ёмкости (например, при приближении руки), а снижение — её уменьшению. Такой формат данных удобен тем, что не требует пересчёта в физические единицы (фарады), а используется для сравнения с базовой линией (baseline) и последующей обработки алгоритмами фильтрации.

Таким образом, являясь надёжным и эффективным инструментом для измерения ёмкостных изменений, библиотека *CapacitiveSensor* позволила в настоящей работе реализовать основной функционал сенсорной панели с минимальными аппаратными затратами.

Фильтрация и буферизация данных. Проблема обеспечение устойчивости показаний ёмкостных сенсоров к помехам в настоящей работе решалась комплексно в сочетании аппаратных методов экранирования с программными алгоритмами фильтрации и буферизации данных.

Алгоритмическая фильтрация была реализована по многоуровневой схеме:

1. Аппаратное усреднение на уровне библиотеки: выполнение методом *capacitiveSensor(10)* 10 последовательных измерений за один вызов с последующим усреднением результата, что эффективно подавляет случайные выбросы на самом низком уровне.

2. Статический порог срабатывания с индивидуальным гистерезисом. После калибровки для каждого сенсора устанавливается персональный порог срабатывания. Величина гистерезиса (40 единиц) была использована на этапе разработки для компенсации характерного уровня шумов наиболее стабильного сенсора.

3. Циклическая буферизация опроса сенсоров с фиксированным интервалом 50 мс использовалась в разрабатываемом ПО для обеспечения предсказуемой нагрузки на процессор и радиоканал, исключения взаимного влияния последовательных измерений и снижения потребления энергии по сравнению с непрерывным опросом.

Адаптация к различной чувствительности сенсоров потребовала разработки гибкой системы анализа, для чего в алгоритм была заложена возможность индивидуальной настройки параметров для каждого канала:

Встраиваемое ПО приёмника. Программное обеспечение приёмной стороны отвечает за приём сигналов с сенсоров, их интерпретацию, распознавание жестов на основе временных комбинаций и управление исполнительными устройствами. Алгоритм реализован для платформы Arduino Uno с использованием библиотеки *RF24.h* для работы с радиомодулем nRF24L01+ [3, 5].

Аппаратная основа приёмника представляет собой связку из платы Arduino Uno, радиомодуля nRF24L01 для приёма данных и RGB-светодиода в качестве исполнительного устройства для визуализации распознанных команд. Светодиод подключён через ШИМ-выводы (пины 3, 5, 6), что позволяет программно формировать любой цвет свечения путём широтно-импульсной модуляции (ШИМ) сигнала.

Программная архитектура построена вокруг концепции конечного автомата (state machine), главными состояниями которого являются:

– Ожидание команды (IDLE) — состояние покоя, в котором система прослушивает эфир.
– Запись сигналов (RECORDING) — состояние, активируемое при получении данных.
– Анализ буфера (PROCESSING) — состояние, в котором сигналы проверяются на превышение пороговых значений, интерпретируются, и осуществляется разбор накопленной последовательности и выполнение соответствующего действия.

Реализация данной логики потребовала объявления следующего набора переменных, каждая из которых выполняет строго определённую роль:

```
#include <SPI.h>                    // Для работы с аппаратным интерфейсом SPI
#include <RF24.h>                   // Для управления радиомодулем
RF24 radio(9, 10);                  // Создание объекта радио: CE на пин 9, CSN на пин 10
const byte address[6] = "00001";    // Уникальный адрес канала для связи с передатчиком

int redPin = 6, greenPin = 5, bluePin = 3;      // Пины ШИМ для подключения каналов R, G, B
int red = 0, green = 0, blue = 0;           // Текущие значения яркости для каждого канала (0-255)
String inputBuffer = "";                    // Динамическая строка для накопления входящих команд
unsigned long lastReceiveTime = 0;    // Временная метка последней полученной команды
const unsigned long comboTimeout = 300;     // Временное окно (мс) для формирования //комбинации

char lastKey = 0;                           // Идентификатор последнего принятого сенсора
unsigned long lastKeyTime = 0;              // Временная метка последнего принятого сигнала
const unsigned long debounceTime = 50;      // Временной интервал (мс), в течение которого
                                    // повторные сигналы от одного сенсора игнорируются

bool blinkMode = false;                     // Флаг активного режима мигания
unsigned long blinkStartTime;               // Время активации режима мигания
const unsigned long blinkInterval = 300;    // Интервал (мс) между сменой состояний
// в режиме мигания
```

Таким образом, программная архитектура приёмника основана на чётком разделении функций между переменными, что обеспечивает надёжную работу системы распознавания жестов, фильтрации помех и управления исполнительными устройствами.

Ключевой особенностью системы является алгоритм распознавания не только одиночных нажатий, но и их комбинаций, выполняемых в быстрой последовательности. Для этого реализован механизм буферизации с плавающим таймаутом.

Функция *processBuffer()* выполняет сопоставление содержимого буфера с заложенной в систему логикой управления. В данном проекте управление реализовано для RGB-светодиода, который реагирует на команды изменением цвета и режимами работы (постоянное свечение, мигание).

Реализация режима мигания обеспечивает визуальную индикацию особых состояний системы. Данный режим активируется при распознавании определённых комбинаций ("12" или “21”) и работает по принципу циклического переключения состояний по истечении заданного интервала времени (*blinkInterval*). В основном цикле *loop()* постоянно проверяется условие: если флаг *blinkMode* установлен в *true*, то система сравнивает текущее время, прошедшее с момента активации режима (*millis() - blinkStartTime*), с заданным интервалом. При превышении интервала состояние светодиода инвертируется (происходит переключение с красного цвета на синий), а время последнего переключения обновляется. Это создаёт периодический эффект мигания, который продолжается до тех пор, пока не будет распознана другая команда.

Таким образом, программное обеспечение приёмной стороны представляет собой гибкую систему, способную не только детектировать факт касания, но и анализировать сложные жестовые комбинации, преобразуя их в понятные команды управления.

Кроссплатформенное приложение для мониторинга. Для визуализации данных, поступающих с сенсорной панели, диагностики системы и анализа её работы в реальном времени было разработано специализированное кроссплатформенное приложение на языке Python [3, 6]. Интерфейс приложения интуитивно понятен и разделен на несколько логических областей. Техническая реализация приложения использует библиотеки *pyserial* для связи с Arduino, *tkinter* для создания GUI и *matplotlib* для построения графиков (рисунок 1). Расположенная под графиками информационная панель включает текстовое поле лога событий с временными метками, индикатор текущей распознанной команды и отображение текущих значений порогов для каждого сенсора (рисунок 2)

Многопоточная архитектура обеспечивает плавную работу интерфейса пока ведётся фоновая обработка данных из последовательного порта.

Таким образом, приложение предоставляет комплексный инструмент для визуального мониторинга работы системы, позволяя в реальном времени наблюдать за сигналами сенсоров, процессами калибровки и распознавания жестов, что существенно упрощает отладку и демонстрацию возможностей разработанной сенсорной панели.

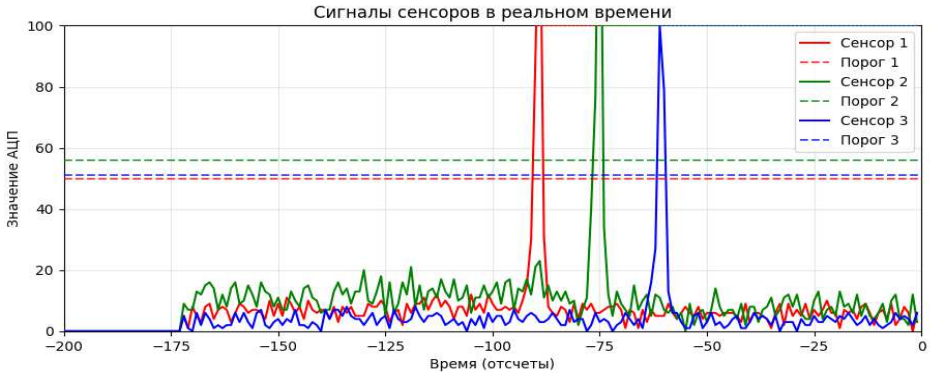

Рисунок 1. – Пример работы графиков приложения

```
12:37:38 - Обнаружено Arduino на порту: COM6
12:37:38 - Доступные порты: COM6
12:37:38 - Программа запущена на Windows
12:37:38 - Для начала работы подключите Arduino и нажмите 'Подключить'
12:37:43 - Подключено к COM6
12:37:43 - Ожидание данных от панели...
12:37:49 - Пороги получены: [48, 56, 51]
12:37:57 - Начало записи команды...
12:37:58 - Выполняется: Действие 13
12:37:58 - Начало записи команды...
12:37:58 - Выполняется: Действие 13
12:37:58 - Начало записи команды...
12:37:58 - Выполняется: Действие 1
12:38:07 - Начало записи команды...
12:38:07 - Выполняется: Действие 2
```

Рисунок 2. – Пример лога событий и распознавания команд

Вывод: Создана система беспроводной передачи данных для управления электронными устройствами, в рамках которой разработанное на базе платформы Arduino Uno встраиваемое программное обеспечение реализует логику управления ёмкостными сенсорами, алгоритм автоматической калибровки и протокол беспроводной связи. Разработанное на языке Python и представленное в настоящей работе кроссплатформенное приложение позволяет осуществлять мониторинг работы панели в реальном времени.

ЛИТЕРАТУРА

1. Емкостные датчики: принцип работы, виды, применение. [Электронный ресурс]. — Режим доступа: https://leuze.ru/emkostnye-datchiki – Дата доступа: 16.09.2025.
2. Кривцов, А. Н. Алгоритмизация и программирование. Основы программирования на C/C++ : учебное пособие / А. Н. Кривцов, С. В. Хорошенко. — Санкт-Петербург : СПбГУТ им. М.А. Бонч-Бруевича, 2020. — 202 с.
3. Петин, В. А. Проекты с использованием контроллера Arduino/ В.А.Петин. - 4-е изд., перераб. и доп. — СПб.: БХВ-Петербург, 2021. — 560 с.
4. Интерфейс передачи данных - SPI [Электронный ресурс] / 3DiY. – Режим доступа: https://3d-diy.ru/wiki/arduino-moduli/interfeys-peredachi-dannykh-spi/ – Дата доступа: 16.09.2025.
5. Подключение к Arduino радиочастотного модуля nRF24L01: схема и программа. [Электронный ресурс]. — Режим доступа: https://microkontroller.ru/arduino-projects/kak-rabotaet-modul-nrf24l01-i-kak-ego-podklyuchit-k-arduino/ – Дата доступа: 16.09.2025.
6. Python и Arduino - их совместное использование с помощью PySerial для управления светодиодом. [Электронный ресурс]. — Дата доступа: https://microkontroller.ru/arduino-projects/ispolzovanie-yazyka-programmirovaniya-python-vmeste-s-arduino/ – Дата доступа: 16.09.2025.