Лабораторная работа № 3.1

Создание системы контроля движения с отслеживанием динамики изменений

Теоретическая часть:

ХАРАКТЕРИСТИКИ HC-SR501

Сегодня мы будем знакомиться с модулем, позволяющим отслеживать движение – пироэлектрическим инфракрасным (PIR) датчиком движения. Для примера будем использовать модуль HC-SR501, как один из самых популярных и совместимых с Arduino. Чаще всего он используется в устройствах, предназначенных для управления освещением, и для этого может использоваться вкупе с датчиком освещённости.

Этот модуль небольшой по размерам, потребляет малый ток и очень простой в использовании, благодаря чему его можно использовать и в устройствах с автономным питанием.

ХАРАКТЕРИСТИКИ ДАТЧИКА:

Широкий диапазон рабочего напряжения: 4,5 – 20 В постоянного тока;

Потребляемый ток покоя: ≈50 мкА;

Напряжение на выходе: 3.3 В;

Рабочая температура: от -15° С до 70° С;

Размеры: 32*24 мм;

Два режима работы;

Максимальный угол обнаружения – 110°;

Максимальная дистанция срабатывания – от 3 до 7 м (регулируется); При температуре более 30° С это расстояние может уменьшаться.

На модуль установлена линза Френеля, которая фокусирует инфракрасные сигналы на пироэлектрический датчик под названием 500BP. Датчик называется PIR (Passive Infra-Red). Пассивный он потому, что для обнаружения движения не используется какая-либо дополнительная энергия, кроме той, что испускается самими объектами.

500BP состоит из двух чувствительных элементов. Управляющая микросхема модуля регистрирует изменения сигналов от обоих элементов и по характеру их изменения обнаруживает движение объектов, испускающих инфракрасные сигналы (живых организмов).

Модуль HC-SR501 имеет 3 вывода:

Питание (VCC);

Земля (GND);

Выход 3v3 (OUT).

Сразу после подачи питания несколько секунд модуль будет калиброваться, в это время возможны ложные срабатывания. Примерно через минуту он перейдёт в режим ожидания. При срабатывании датчика на выходе появляется логическая единица, напряжение – 3.3 вольта.

Изменения этого сигнала зависят от выбранного режима работы. Он меняется перемычкой (отмечена на фото с подписями какой режим будет выбран). Если выбран H– при нескольких срабатываниях подряд на выходе датчика остаётся высокий уровень, при L– для каждого срабатывания будет подан свой импульс.

Также на самом модуле можно найти два переменных резистора, регулирующих дистанцию обнаружения движения (Distance Adjust) и время, в течение которого на выходе будет логическая единица (Delay Time Adjust). Дистанция регулируется в пределах 3 – 7 метров, задержка от 5 до 300 секунд.

И ещё немного о его особенностях. При работе с датчиком следует избегать источников света и тепла, закрывающих поверхность объектива модуля. Ветер также может создавать помехи. На большем расстоянии датчик более чувствителен.

Работа с библиотеками python:

Для работы с COM-портами в Python нам понадобится библиотека **PySerial**, предоставляющая удобный интерфейс для последовательной связи. **PySerial** можно легко установить с помощью pip, менеджера пакетов Python. Откройте терминал или командную строку и выполните следующую команду:

```
pip install pyserial
```

Перед подключением к определенному COM-порту очень важно определить доступные порты в вашей системе. **PySerial** предлагает удобную функцию **serial.tools.list_ports.comports()**, которая возвращает список доступных COM-портов. Давайте посмотрим, как его использовать:

```
import serial.tools.list_ports

ports = serial.tools.list_ports.comports()

for port in ports:
    print(port.device)
```

После того, как вы определили нужный COM-порт, вы можете создать соединение с помощью **PySerial**. Класс **serial.Serial()** предоставляет необходимые методы для настройки и управления подключением. Вот пример:

```
import serial

port = "COM1"  # Replace with the appropriate COM port name
baudrate = 9600  # Replace with the desired baud rate

ser = serial.Serial(port, baudrate=baudrate)

# Perform operations on the COM port

ser.close()  # Remember to close the connection when done
```

Для связи с устройством, подключенным к COM-порту, нам нужно понять, как читать и записывать данные. **PySerial** предлагает для этой цели два основных метода: **ser.read()** и **ser.write()**. Давайте рассмотрим эти методы:

```
# Reading data
data = ser.read(10)  # Read 10 bytes from the COM port
print(data)
```

```
# Writing data
message = b"Hello, world!"  # Data to be sent, should be in bytes
ser.write(message)
```

PySerial предоставляет широкие возможности для настройки COM-портов в соответствии с конкретными требованиями. Вы можете настроить такие параметры, как биты данных, стоповые биты, четность, время ожидания и управление потоком. Вот пример:

```
ser = serial.Serial(port, baudrate=baudrate, bytesize=8, parity='N', stopbits=1, timeout=1, xonxoff=False, rtscts=False)
# Adjust the parameters as needed
```

При работе с COM-портами очень важно корректно обрабатывать ошибки. PySerial вызывает исключения для различных сценариев ошибок, таких как ненайденный порт, отказ в доступе или тайм-ауты связи. Реализация соответствующей обработки ошибок обеспечивает надежность ваших приложений.

Вот пример обработки ошибок при работе с COM-портами в Python с помощью PySerial:

```
import serial
import serial.tools.list_ports

try:
    # Find and open the COM port
    ports = serial.tools.list_ports.comports()
    port = next((p.device for p in ports), None)
    if port is None:
        raise ValueError("No COM port found.")

    ser = serial.Serial(port, baudrate=9600)

    # Perform operations on the COM port
```

```
    ser.close()  # Close the connection when done

except ValueError as ve:
    print("Error:", str(ve))

except serial.SerialException as se:
    print("Serial port error:", str(se))

except Exception as e:
    print("An error occurred:", str(e))
```

В этом примере мы пытаемся найти и открыть COM-порт, используя **serial.tools.list_ports.comports()**. Если COM-порт не найден, возникает ошибка **ValueError**. Если существует исключение последовательного порта, например порт уже используется или недоступен, перехватывается **SerialException**. Любые другие общие исключения перехватываются блоком **Exception**, который обеспечивает сбор всех непредвиденных ошибок. Каждый блок ошибок обрабатывает определенный тип исключения и выводит соответствующее сообщение об ошибке.

Реализуя обработку ошибок таким образом, вы можете изящно обрабатывать различные сценарии ошибок, которые могут возникнуть при работе с COM-портами в Python.

ПРИМЕНЕНИЕ SQLITE

База данных — это набор структурированной информации. Для ее изменения требуются системы управления — СУБД. Как и любая СУБД, SQLite позволяет записывать новую и запрашивать существующую информацию, изменять ее, настраивать доступ.

Благодаря свойствам SQLite применяется:

- на сайтах с низким и средним трафиком;
- в локальных однопользовательских, мобильных приложениях или играх, не предназначенных для масштабирования;
- в программах, которые часто выполняют прямые операции чтения/записи на диск;
- в приложениях для тестирования бизнес-логики.

SQLite не требует администрирования и работает на мобильных устройствах, игровых приставках, телевизорах, беспилотных летательных аппаратах, камерах, автомобильных мультимедийных системах и т.д. СУБД использует множество программ: Firefox, Chrome, Safari, Skype, XnView, AIMP, Dropbox, Viber и другие.

КАК РАБОТАЕТ SQLITE

Большинство СУБД используют клиент-серверную архитектуру: данные хранятся и обрабатываются на сервере, а запросы к нему посылает клиент. «Клиент» — это часть программы, с которой взаимодействует пользователь. «Сервером» может быть и отдельный процесс на том же компьютере (так называемый демон), и стороннее устройство, как в случае с сайтами.

SQLite устроена иначе и не имеет сервера. Это значит, что все данные программное обеспечение хранит на одном устройстве. СУБД встраивается в приложение и работает как его составная часть. Если установить на компьютер программу, использующую SQLite, то база данных тоже будет храниться на нем же. Формат базы — один текстовый файл, который можно прочитать на любой платформе. Такой подход повышает производительность и скорость работы.

Работать с SQLite можно как с библиотекой или через SQLite3.

ЧТО ТАКОЕ SQLITE3

SQLite3 — это консольная утилита для работы с SQLite от разработчиков СУБД. Она запускается и работает в командной строке, в консоли операционной системы. Можно скачать версии для Windows, Mac OS и Linux.

По функциональности SQLite3 — программа-клиент для клиент-серверных приложений. С ее помощью можно вводить и передавать запросы к базе данных: создавать, модифицировать, получать или удалять таблицу. Разница в том, что она обращается не к отдельному процессу-серверу, а ко встроенному в приложение движку SQLite.

В SQLite3 можно писать SQL-код: утилита отправит запрос к ядру, получит и отобразит результат.

ПРЕИМУЩЕСТВА SQLITE

Высокая скорость. Благодаря особенностям архитектуры SQLite работает быстро, особенно на чтение. Компоненты СУБД встроены в приложение и вызываются в том же процессе. Поэтому доступ к ним быстрее, чем при взаимодействии между разными процессами.

Хранение данных в одном файле. База данных состоит из табличных записей, связей между ними, индексов и других компонентов. В SQLite они хранятся в едином файле (database file), который находится на том же устройстве, что и программа. Чтобы при работе не возникало ошибок, файл блокируется для сторонних процессов перед записью. Раньше это приводило к тому, что записывать данные в базу мог только один процесс единовременно. Но в новых версиях это решается перенастройкой режима работы СУБД.

Минимализм. Создатели SQLite пользуются принципом «минимального полного набора». Из всех возможностей SQL в ней есть наиболее нужные. Поэтому SQLite отличают малый размер, простота решений и легкость администрирования. Для повышения базовой функциональности можно использовать стороннее программное обеспечение и расширения.

Надежность. Код на 100% покрыт тестами. Это означает, что протестирован каждый компонент ПО. Поэтому SQLite считается надежной СУБД с минимальным риском непредсказуемого поведения.

Нулевая конфигурация. Перед использованием СУБД не нужна сложная настройка или длительная установка. Для решения большинства задач ей можно пользоваться «из коробки», без установки дополнительных компонентов.

Малый размер. Полностью сконфигурированный SQLite со всеми настройками занимает меньше 400 Кб. Если использовать СУБД без дополнительных компонентов, размер можно уменьшить до 250 Кб. Он зависит только от количества загруженной информации. Несмотря на малый размер, SQLite поддерживает большинство функций стандарта SQL2 и имеет ряд собственных.

Доступность. SQLite находится в публичном доступе. На ее использование нет правовых ограничений, а владельцем считается сообщество. Можно открывать, просматривать и изменять исходный код установленного ПО.

Кроссплатформенность. СУБД подходит для UNIX-подобных систем, MacOS и Windows.

Автономность. Система независима от стороннего ПО, библиотек или фреймворков. Чтобы приложение с базой на SQLite работало, дополнительные компоненты не требуются. Также не обязателен доступ в интернет: вся база хранится на устройстве, получить данные можно локально.

НЕДОСТАТКИ SQLITE

Ограниченная поддержка типов данных. SQLite поддерживает только четыре типа данных, которые реализованы в SQL:

INTEGER — целое число;

REAL — дробное число;

TEXT — текст;

BLOB — двоичные данные.

Также существует особое значение NULL — отсутствие данных.

Отсутствие хранимых процедур. Так называются блоки кода на SQL, которые сохраняются в базу данных. Хранимые процедуры можно вызывать как отдельные функции, и это удобно, если нужно последовательно выполнить несколько однотипных действий. Но SQLite их не поддерживает из-за особенностей архитектуры.

Ограничения в применении. Отсутствие сервера — преимущество и недостаток одновременно. Без сервера возможности СУБД меньше. Например, к одной базе не смогут обращаться несколько разных устройств.

В SQLite ограничена многопоточность — единовременное выполнение нескольких процессов. Одновременно читать из базы могут несколько процессов, а писать в нее по умолчанию — только один. В версии 3.7.0 в SQLite внедрили возможность записи разными приложениями, но даже так она уступает клиент-серверным СУБД по возможностям работы с потоками. Поэтому SQLite не подойдет для многопользовательских приложений или программ, записывающих большой объем данных.

Отсутствие бесплатной техподдержки. Стоимость профессиональной технической поддержки от разработчиков — от $1500 в год. Чтобы получить информацию бесплатно, потребуется пользоваться форумами и руководствами от пользователей, а также официальной документацией.

Отсутствие встроенной поддержки Unicode. Unicode — это популярный стандарт кодирования символов. Он включает практически все существующие знаки и буквы, поэтому считается самым распространенным в мире. Без его поддержки приложение не сможет корректно работать с кириллицей, иероглифами и многими другими символами. SQLite «из коробки» не поддерживает Unicode, его нужно настраивать отдельно. Это может вызвать сложности с локализацией.

Практическая часть:

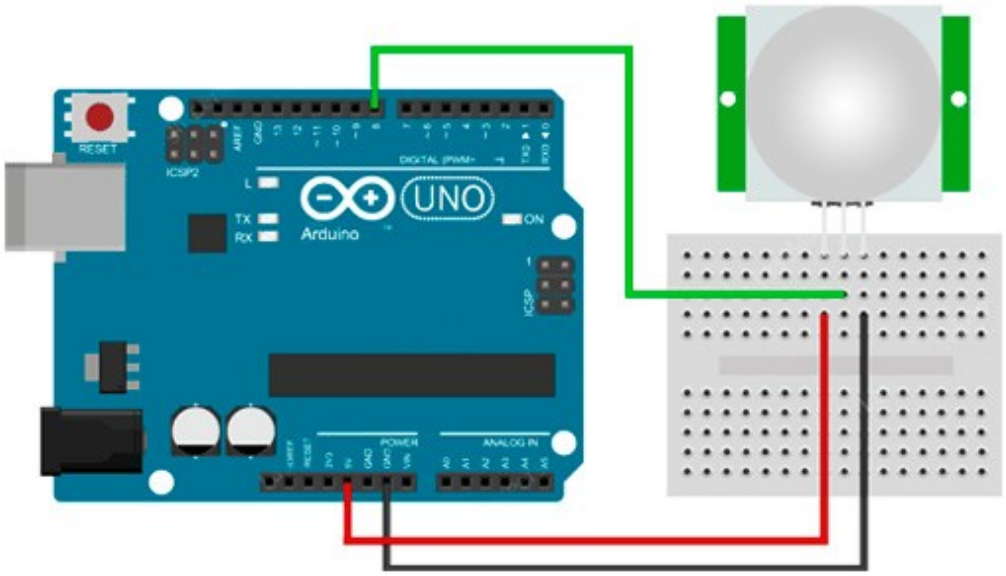
Подключение датчика движения HC-SR501 к Arduino

Итак, мы прошли теорию, настало время проверить датчик в работе. Первым этапом станет его подключение к Arduino:

- **GND** подключаем к одноимённому выводу Arduino;
- **VCC** к **5V**;
- **OUT** подключим к **A0**.

Подключение выхода модуля к аналоговому пину связано с тем, что цифровые пины Arduino работают с пятивольтовой логикой, а наш модуль рассчитан на 3.3 В. В скетче за логическую единицу мы примем значение более 500 на аналоговом порту. Это будет около 2.44 В и более.

Схема подключения HC-SR501:



Соответствующий код для подключения и отображения данных

```
#define pirPin 8
#define LedPin 13

void setup() {
  Serial.begin(9600); // Объявляем работу com порта со скоростью
9600
  pinMode(pirPin, INPUT); //Объявляем пин, к которому подключен
датчик движения, входом
}

void loop() {
  int pirVal = digitalRead(pirPin); //Считываем значения с
датчика движения. Если обнаружили движение,
                                    //то транслируем сигнал
тревоги в монитор порта и включаем светодиод
  if(pirVal == HIGH)
  {
        Serial.print("Тревога ");
    delay(2000);
  }
else
{
  Serial.print("Сканирую ");
  delay(1000);
}
}
```

Подключение схемы к python

Сначала необходимо установить библиотеки для подключения схемы к Arduino:

```
python -m pip install pyserial
```

Далее произведем подключение нашей схемы к python через соответствующий порт:

```
import serial

ser = serial.Serial('#ваш порт', 9600) # Подключаемся к com порту на скорости 9600
pirPin = 8

while True:
  pirVal = ser.readline().decode('utf-8').rstrip() # Считываем значения с датчика движения
  if pirVal == "HIGH": # Если обнаружили движение, то выводим сообщение о тревоге
    print("Тревога")
  else:
    print("Сканирую") # В противном случае выводим сообщение о сканировании
  ser.flushInput() # Очищаем буфер входного потока данных, чтобы избежать ошибок при
чтении следующих данных.
```

Создание и подключение схемы к базе данных sqllite3

```
import os
import sqlite3

# Create DB in current file
DEFAULT_PATH = os.path.join(os.path.dirname(__file__),
'StudentDB.db')
CREATE_SQL_FILE = os.path.join(os.path.dirname(__file__),
'ValueDB.sql')


def db_connect(db_path=DEFAULT_PATH):
    con = sqlite3.connect(db_path)
    return con


def create_table():
    db_conn = db_connect()

    with db_conn:
        try:
            db_conn = db_connect()
            cursor = db_conn.cursor()
            print("Successfully Connected to SQLite")

            with open(CREATE_SQL_FILE, 'r') as sqlite_file:
                sql_script = sqlite_file.read()

            cursor.executescript(sql_script)
            print("SQLite script executed successfully")
            cursor.close()

        except sqlite3.Error as error:
            print("Error while executing sqlite script", error)

    print("Successfully created table!")


def create_student_task(conn, table123):
    sql = ''' INSERT INTO TABLE123(id, value)
              VALUES(?,?,?) '''
    cur = conn.cursor()
    cur.execute(sql, table123)
    conn.commit()
    return cur.lastrowid


def create_values(id, pirVal):
    # create a database connection
    db_conn = db_connect()

    with db_conn:
        value1 = (id, pirVal)
```

```
        # create student
        create_student_task(db_conn, value1)



def main():
    create_table()
    create_values(1, ‘HEllo’)


if __name__ == '__main__':
    main()
```

Итоговый код на питоне будет иметь вид:

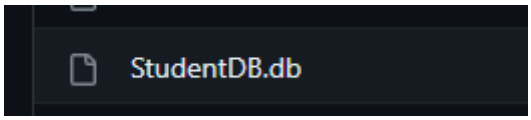
```
import os
import sqlite3
import serial


# Create DB in current file
DEFAULT_PATH = os.path.join(os.path.dirname(__file__),
'StudentDB.db')
CREATE_SQL_FILE = os.path.join(os.path.dirname(__file__),
'ValueDB.sql')


def db_connect(db_path=DEFAULT_PATH):
    con = sqlite3.connect(db_path)
    return con


def create_table():
    db_conn = db_connect()

    with db_conn:
        try:
            db_conn = db_connect()
            cursor = db_conn.cursor()
            print("Successfully Connected to SQLite")

            with open(CREATE_SQL_FILE, 'r') as sqlite_file:
                sql_script = sqlite_file.read()

            cursor.executescript(sql_script)
            print("SQLite script executed successfully")
            cursor.close()

        except sqlite3.Error as error:
            print("Error while executing sqlite script", error)
```

```
    print("Successfully created table!")


def create_student_task(conn, table123):
    sql = ''' INSERT INTO TABLE123(id, value)
              VALUES(?,?,?) '''
    cur = conn.cursor()
    cur.execute(sql, table123)
    conn.commit()
    return cur.lastrowid


def create_values(id, pirVal):
    # create a database connection
    db_conn = db_connect()

    with db_conn:
        value1 = (id, pirVal)
        # create student
        create_student_task(db_conn, value1)



def main():
    create_table()
      a = 1
      ser = serial.Serial('#ваш порт', 9600) # Подключаемся к com порту на скорости 9600
      pirPin = 8

      while True:
      pirVal = ser.readline().decode('utf-8').rstrip() # Считываем значения с датчика
движения
      create_values(a, pirVal)
      a += 1
      if pirVal == "HIGH": # Если обнаружили движение, то выводим сообщение о
тревоге
      print("Тревога")
      else:
      print("Сканирую") # В противном случае выводим сообщение о сканировании
      ser.flushInput() # Очищаем буфер входного потока данных, чтобы избежать
ошибок при чтении следующих данных.



if __name__ == '__main__':
    main()
```

SQL script ValueDB.sql

```
CREATE TABLE TABLE123 (
 id INTEGER PRIMARY KEY,
 value INT NOT NULL
);
```

В результате вы должны увидеть созданную базу данных:



Так же в базе данных будет создана 1 запись, которую вы завели руками

Усложняем задачу:

1) Создайте автоматическую подгрузку данных с датчика в базу данных.
2) Сделайте дополнительное логирование времени записи и записывайте его в базу данных рядом со значением. Для этого нужно создать дополнительную колонку.
3) Добавьте в схему сервопривод, и заставляйте его поворачиваться, каждый раз, когда рядом происходит движение
4) Произвести сохранение данных в csv файл в дополнение к легированию в базе данных.

   ***

   5) Построить динамическую гистограмму соотношения открытого и закрытого состояния двери из пункта 2. Вывести итог в виде графика matplotlib.