FirstKlaas Documentation

Release 0.9.1

Klaas Nebuhr

Mar 02, 2020

Contents

CHAPTER 1

An interactive LED matrix based on WS2812b LEDs

1.1 The Idea

A LED Matrix which can be used as a display to play games or to show different information (like the current weather conditions)

1.2 The Architecture

The main ingredients are:

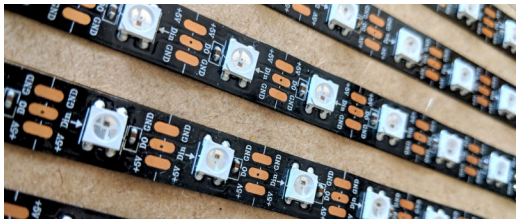
- Raspberry PI (Version 2 or 3 will do)
- Arduino Uno (for wireless panels use a NodeMCU)
- LW2812b LED Stripes

1.3 Python

For easy color manipulation as well as different hardware setups. I wrote a python library. To get started, read the documentation of the `Display` class.

1.4 Contents

1.4.1 Getting started



The LEDs

Ok, this is a library to manipulate leds. So, we definitly need leds. The arduino sketch makes use of the fastled library. I only used WS2812B led stripes, but any led type supported by the library should work. If you're using any other led, you may need to tweak the sketch to initialize fastled correctly. For more details on fastled it is worth checking out the website Fastled.io. There you will also find some useful information about timings and power consumtion.

The micro controller

If you clone the github project, you will find two arduino/nodemcu sketches. The correct sketch depends on the way you want to communicate with the controller. A sketch counterparts a certain sender class. Currently two sketches are available. One for the `UDPSender` class and one for the `SerialSender` class.

Here I will focus on the serial communication. The sketch you need is the `Framebuffer3.ino` sketch.

```
#include <Arduino.h>
#include <FastLED.h>

#define DATA_PIN              5     // The Pin your LED data line is connected to
#define BAUDRATE         500000     // The communication speed
#define MAX_SERIAL_DELAY   3000     // Maximum delay between transmitted
                                    // data within one command in milliseconds
const uint8_t numberOfLeds  = 100;  // The number of leds

// ### DO NOT CHANGE ANYTHING BELOW THIS LINE UNLESS YOU KNOW WAHT YOU ARE DOING ###

#define BYTES_PER_PIXEL      3   // Don't chang this
#define CMD_PAINT_PANEL    243   // Don't chang this

#define NODEBUG

const int16_t numberOfByte = numberOfLeds * BYTES_PER_PIXEL;
```

Before uploading the script, some adjustments have to be done to match your setup.

DATA_PIN The pin on the micro controller, where the data line of your leds is connected to. The default is pin 5.

BAUDRATE The baudrate for the serial communication. The baudrate in the sketch and the baudrate in the constructor of the SerialSender have to be identical. Please read the documentation of the pyserial module, as not all speeds are supported on every plattform. Some people reported, that you have to change boot settings on the raspberry before using higher speeds than 155,200 b/s. On my raspberry 3 the default speed of 0,5 Mb/s worked perfectly. Keep in mind, that also the quality and the length of the usb cable will have an impact.

numberOfLeds The variable `numberOfLeds` holds (surprise, surprise) the number of LEDs on your panel.

> **Warning:** Make shure you don't use more leds in your python `Display` instance. Also make shure that your physical board doesn't have less leds.

Installing the python library

The library is hosted on PyPi. You can easily install it using pip. The library only suppports python3. The following shell session shows the process to install the library. The procedure shown below works on ubuntu and raspberry. I've never tested this on windows. But installing on windows should be pretty straight forward.

```
$ mkdir tmp
$ cd tmp
$ python3 -m venv .
$ . bin/activate
(tmp) $ pip install ledwall
```

The library has the following dependencies:

- pyserial
- paho-mqtt
- inputs

The `Display` class offers some functions that depend on the python image library PIL. PIL is not supported on python3. Therefore PIL is not listed in the dependencies and will not be installed automatically. You can instead install the pillow library if you want to use these functions. Maybe this will change in the future.

Writing your first script

You can fire up the python3 REPL console or use your favorite editor or ide.

> **Tip:** The library supports only python 3. Scripts will not run with python 2.

The Hello World of LEDWall

```
import ledwall.components as comp

s = comp.SerialSender()             # Creating a serial sender with the
                                    # default port_name and baudrate
d = comp.Display(16,32,s)           # Defining a new display component

red   = comp.Color(255,0,0)         # Defining an RGB color
col   = comp.HSVColor(0.7,0.8,1.0)  # Defining an HSVColor

d.fill((0,255,0))                   # Fill the panel with green. You can specify
                                    # rgb colors as tuple as you go
d.set_pixel(0,3,red)                # which is equivalent to:
```

(continues on next page)

(continued from previous page)

```
                                    # d[(0,3)] = red or d[(0,3)] = (255,0,0)
d.set_pixel(14,23,col)              # Setting the pixel to the defined
                                    # hsv color. Comversion to rgb is calculated
                                    # on the fly

col.hue += 0.13                     # Changing the HUE component of the color
d.set_pixel(15,23,col)              # Setting the pixel in this color

d.update()                          # Updating the physical component
```

1.4.2 Python Library

The python library provides modules and classes to manipulate the pixel colors on the LED Display. I tried to design a pythonic API, which makes it very intuitive to paint to the LED Display.

A very simple python script would look like this:

```
from ledwall.components import *

# Create a new display instance. Using a SerialSender to
# send the color data to the arduino.
# Setting the desired framerate is 15
d = Display(16,32, SerialSender(portName='/dev/ttyACM0', baudrate=1000000),␣
↪framerate=15)

# Defining a few basic colors
red   = RGBColor.fromIntValues(255,0,0)
green = RGBColor.fromIntValues(0,255,0)

d.fill(green)
d.setPixel(0,3,red)
d.setPixel(14,23,red)

d.update()
```

ledwall.components

The components module is the home for the Display class, which is the main class of this library. A display is a in memory representation of the color state of every led. It supports different cabeling schemes for your hardware component and many convenient method to control the pixels of the display.

Also this modules offers color classes for the RGB and the HSV color space.

WireMode

Display

Color Classes

For a LED project color manipulation is of course the most important part. The WS2812b needs for every pixel the RGB values, where every component is represented as byte [0;255]. From a artistic point of view the HSV color space

is much more intuitive. Therefore this library provides convenience classes to create, manipulate and convert classes in RGB (`RGBColor`) and HSV (`HSVColor`) color space.

For more information about the HSV color space, checkout the wikipedia page.

Gamma Correction

The library uses the following table for gamma correction. Each channel value from an RGB color is map to the corresponding corrected color. This table is especially made for the WS2812b leds. To convert a byte u can use the static method `gammaCorrection()`.

```
gamma8_table = [  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,
                  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  0,  1,  1,  1,  1,
                  1,  1,  1,  1,  1,  1,  1,  1,  1,  2,  2,  2,  2,  2,  2,  2,
                  2,  3,  3,  3,  3,  3,  3,  3,  4,  4,  4,  4,  4,  5,  5,  5,
                  5,  6,  6,  6,  6,  7,  7,  7,  7,  8,  8,  8,  9,  9,  9, 10,
                 10, 10, 11, 11, 11, 12, 12, 13, 13, 13, 14, 14, 15, 15, 16, 16,
                 17, 17, 18, 18, 19, 19, 20, 20, 21, 21, 22, 22, 23, 24, 24, 25,
                 25, 26, 27, 27, 28, 29, 29, 30, 31, 32, 32, 33, 34, 35, 35, 36,
                 37, 38, 39, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 50,
                 51, 52, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 66, 67, 68,
                 69, 70, 72, 73, 74, 75, 77, 78, 79, 81, 82, 83, 85, 86, 87, 89,
                 90, 92, 93, 95, 96, 98, 99,101,102,104,105,107,109,110,112,114,
                115,117,119,120,122,124,126,127,129,131,133,135,137,138,140,142,
                144,146,148,150,152,154,156,158,160,162,164,167,169,171,173,175,
                177,180,182,184,186,189,191,193,196,198,200,203,205,208,210,213,
                215,218,220,223,225,228,231,233,236,239,241,244,247,249,252,255 ]
```

Color

HSVColor

RGBColor

Sender

SerialSender

MqttSender

UDPSender

ConsoleSender

This class was used during development. It is documented for the sake of completeness. There is no real use case. So, save time and stop reading.

RegionSender

ledwall.geometry

The geometry module includes some basic shapes that can be drawn to the display. The basic shapes are `Point`, `Line`, `Rectangle`.

Some sample code:

```
from ledwall.geometry import *

p1 = Point(2,4)
p2 = Point(7,12)

# Line between two points
l1 = Line(p1,p2)

# Line from values (x1,y1,x2,y2)
l1 = Line.fromTuple((0,0,5,10))

r1 = Rectangle(2,3,10,20)
```

Point

Rectangle

Line

1.5 Indices and tables

- genindex
- modindex
- search