**Winter has come, time to get cosy**

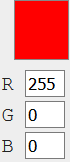
Did someone leave open the fridge door? Get the heaters running, I’m freezing!   
Winter has finally arrived in Konstanz, just when we were dealing with the colour warmth of our Ambilight. We guess that’s the perfect motivation to puzzle out a neat algorithm, which adapts the colour set in the lighting profile to the temperature of the environment. Let’s get it cosy in here, shall we?

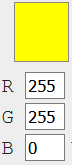
For this project, we’re using a Neopixels RGB LED strip by Adafruit. To control the LEDs, we use the Neopixel library for Arduino. The function, which sets the LEDs to a certain colour expects RGB values. For everyone, who is not familiar with the RGB colour space, let us quickly give you a crash course:

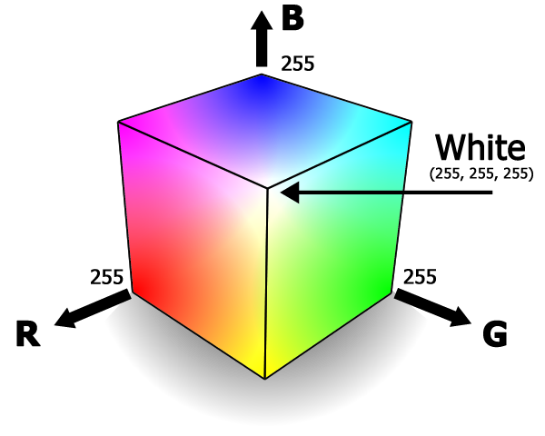
**The RGB colour space**

RGB stands for the three colours chromaticities **R**ed, **G**reen and **B**lue. The RGB colour space basically includes all colours, which can be made by mixing three coloured lights, one light for red, one for green and one for blue. When the red component is set to 0, the LED is turned off, when it is set to 255, it is fully turned on. With any value between, the LED will be on partial light emission. So if all three components use 8 bits (0-255), the representable colour space includes

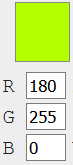
256\*256\*256 = 16777216 different colours.

For example: If the red light is on and the other two are out, you will obviously get red. This colour will have the RGB value (255,0,0) or #FF0000 in the hexadecimal representation. 

If you turn on the red and the green light, you will get a rich yellow with the RGB value (255,255,0) and #FFFF00 respectively. 

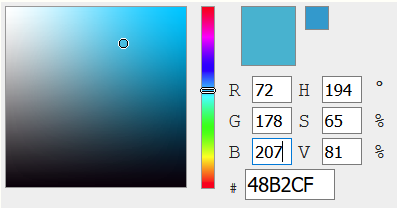


[Source: <http://www.drmoron.org/images/is-black-a-color/color-cube.png>]

By reducing the intensity of the red light, ergo reducing the value of the red component to, let’s say to 180, the green part will become dominant and the result colour will become a light green, with the RGB value of (180,255,0) and #B4FF00 respectively. 

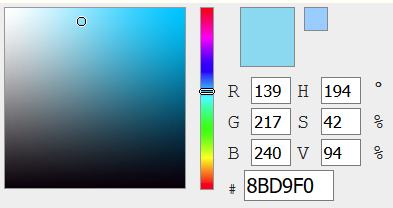
**You’re hot and you’re cold**

Now back to our project: On the Android App, the user can set the colour of the Ambilight. Originally, we intended, that one will be able to pick from the entire RGB colour space, but we found this would make the colour warmth adjustment significantly more complex. If we wanted to achieve the changes of warmth simply by changing the saturation of the given colour, this wouldn’t be a problem at all. But then an intense blue (0,0,255) couldn’t get any warmer, even though the colour is a rather cold one. Therefore we need to let the colour “approach” a lilac kind of blue to achieve warmer temperatures, meaning we want to not only change the saturation, but also the colour shade.  
Let’s assume the user picks a colour, which is composed with all 3 RGB components:



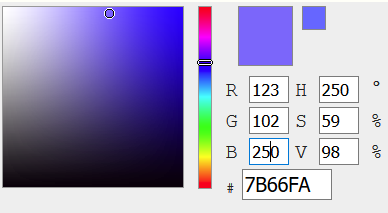
Firstly, we would have to identify the dominant colour component, which here would be blue. This would have to stay dominant, so the new colour would still be in the same “colour family”.

An adjustment to a cooler colour could look like this:



This adjustment would require changes in all 3 RGB values, with a big increase in red and approximately the same increase in both green and blue.

Respectively, an adjustment to a warmer colour could look like the following:



This also requires changes in all three RGB values, with increases of blue and red and a significant decrease of green.

With other colours however, the required adjustments could look little up or even entirely different, because they naturally may have another warmth to begin with. Let’s take a look at a green colour for example:

[orig]

A cooler version of this colour could look like this for example:

[cooler2]

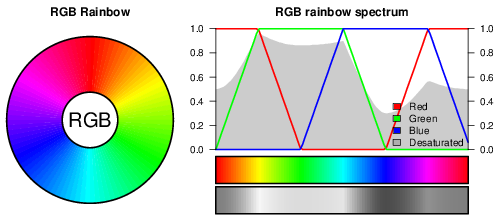
And a warmer one like this:

[warmer2]

So if we have to always take all 3 RGB values into consideration, when trying to categorize a given colour in order to find out how to adjust them, things would just get pretty complicated.

**Let’s get the pot of gold at the end of the rainbow**

What we did to avoid this, we reduced the available colour space to the so-called rainbow colours.



We expect most users to mostly choose saturated colours, so this is an acceptable trade off – it reduces the development effort and doesn’t take away much from the user. If one RGB value is always 0 and one other always 255, we can categorize the colour shade very easily and know which adjustments to make. This can be implemented with a Switch-Case.

**What’s next?**

Well, we already set up a table of colour “categories” and the respective strategies, now we must implement the Switch-Case statement and balance out all the parameters of the adjustment algorithm. How sensitive should the controller react to temperature changes? How much do we want to use the saturation to create warm or cool colours? You get the drift…

When that part is done, all there is left is setting everything together and testing everything out. Let’s get busy then ;)