# RGBSensor.h

This file talks to the RGBSensor (tcs34725) module to detect certain colors. We use this to figure out if we won the game by detecting the color red.

If you don’t know what RGB means or what its values represent, start reading here:   
<https://geraldbakker.nl/psnumbers/rgb-explained.html>

Text

Description automatically generated with medium confidence

The official library (Adafruit\_tcs34725) uses a delay on the color detector to determine the color. This blocks the main loop made it so the other things the robot have to keep on doing are not happing. For this reason we are using the “tcs34725” library, which allows us to run the RGBSensor without a delay.

We initialize the TCS34725 library instance on the “useRGBSensor” variable.

Text, letter

Description automatically generated

This function is used to detect the color red.

We first define a int array that allows 3 values ([red, green, blue]).

The “colorRedDetected()” function returns a boolean (true, false) based on whether the detected color was red or not.

Some manual testing was done to determine the right values for the colored papers we use. The module has a LED light that can morph the color somewhat (red can become pink for example).

The values that work consistently are:

* Red more then 200
* Green less then 40
* Blue less then 40

When all of this is the case, we return true and stop the function for executing further.

If it’s not the case, we return false.

Text

Description automatically generated

This is the function the will read the values from the module and set those in the RGBValues variable we defined before.  
This function runs every 1s (1000ms) using the timer. Because of that the function has to return a boolean and has to except a “void\*” type parameter.

The “useRGBSensor.updateData()” function will try to update the RGB values inside of the library. We pass “true” to this function as a parameter to tell the library to do this without a delay (without blocking the main loop). If this succeeded, and the colors are updated correctly it will return true.

This value is then stored in the “RGBStatus” variable.

When RGBStatus is true, and the RGB values where successfully updated, we will have to calculate the correct RGB values. This is because the raw value for each color returns 0-1023 instead of 0-255. I have followed the example given in this code:  
<https://github.com/jorgemvc/SensorRGB/blob/master/SensorRGB.cpp>

An explanation of what is happening for the color red: (it’s the same for all 3 colors).

We define a float (decimal number) variable called “red”. We then assign the raw red value to it coming from the library (useRGBSensor.r\_raw).

We then divide this number with the raw value for “c”. “c” is the “color” value coming from the library which is also between 0-1023. It tells us how much of a color ratio has been detected (between black/white -> color).

By dividing the raw red value by the raw color value we will get back a decimal number.  
An example:  
400 (raw red) / 800 (raw color) = 0.5  
If we then multiply this by 256 (0-255) we will get 128 as the red RGB value.

After all this, we will assign the calculated values to the RGBValues array indexes [0 for red, 1 for green, 2 for blue).  
Since all the variables (red, green, blue) are floats, in some edge cases we can get a value higher then 255, and it will also always hold a decimal (120 would be 120.0).

If red is more than 255, we set RGBValues[0] to 255. Else we will set RGBValues[0] to “floor(red)”. This will round down numbers to an int (e.g. 244.5 will become 244).

When we do this for all the values, we now end up with correct RGB color values.

At the end of the function we return “true” to tell the timer it can run this function again.

Graphical user interface, text, application, email

Description automatically generated

This code is run when the Arduino starts up. It initializes the RGB sensor.

If useRGBSensor.begin() returns false, it means it was not able to detect the module, if this is the case we will print an error to the serial monitor in the Arduino IDE and stop the code from executing further.

When the sensor is detected, we will write some predefined variables form the library to the module directly. This helps the module to automatically interrupt the RGB sensor to detect the colors correctly even if we are not using the interrupt PIN on the module.

This is taken from the example here:  
<https://github.com/adafruit/Adafruit_TCS34725/blob/master/examples/interrupt/interrupt.ino>

The interrupt is needed so the module stops detecting the RGB values for a while, allowing us to read the values. The TCS34725\_PERS and TCS34725\_PERS\_NONE variables are defined and used from the library itself (I am not sure what the values are).

When this is all done, we will start the timer that runs the “setRGB()” function every 1000ms (1s).

# LedControl.h

Text

Description automatically generated with medium confidence

W include the FastLED library, it is a powerful library which allows us to control LED lights and is also used on the timer to control the LED strip. The code in this file is for controlling the tiny white led light inserted directly on the smart car shield (the hub sitting at the very top, connected to the Arduino board).

The 3 variables (LED\_PIN, LED\_TOTAL, LED\_BRIGHTNESS) are created to connect to 3 different pins on the Arduino board. E.g. LED\_PIN is a variable connected to pin 4, so whenever we call this variable, then we can tell Arduino what happens on pin 4.

CRGB leds[LED\_TOTAL] initializes the FastLED library instance on the “leds” variable, here we pass to total amount of LEDS we want to control. It will return an array with all the available LED lights, which will always hold 1 item in this case (leds[0]).

Text

Description automatically generated

We create a ledLight variable that holds the active color, this variable can be used in other places in our code to determine which color the LED light currently is. This is used in the “Bluetooth.h” file. By default we set this to “off” since when the Arduino starts up no light is being shown.

The changeLED function allows use to change the color, it except the color we want to set as a parameter (color). We accept 6 colors:

* Red
* Blue
* Orange
* Green
* Black (or off, same thing)
* Purple

When the color we pass is red, we will set the first led light on the CRGB instance (leds[0]) to the color red using “CRGB::Red”.

This is simply how this library works as seen in the documentation here:  
<https://github.com/FastLED/FastLED/wiki/Basic-usage>

When the passed color is set to the LED, we will update the “ledLight” variable to match this color.

After that we trigger “FastLED.show()”, which will actually output the color to the LED light so it is visible.

A picture containing logo

Description automatically generated

This function is run when the Arduino start and initializes the FastLED library.

We use the “FastLED.addLeds()” function which accepts the “leds” array instance we create before and the “LED\_TOTAL” variable holding the amount of LED lights we need to control.

The “<NEOPIXEL, LED\_PIN>” tells the FastLED library what type of LED light we are using, which in this case is a NEOPIXEL led (It’s just the name of the LED type). We also tell it on which pin the LED us communicating with the Arduino, so its able to send the color data to the LED.

When this setup is done, we will set the brightness of the LED using the “FastLED.setBrightness()” function, we pass in the LED\_BRIGHTNESS variable which is set at the start of the file to 20. This value is a ratio between 0 – 255.

At the end we will trigger the “changeLED()” function mentioned before and set it to “off”. This makes sure the light will be turn off initially, just in case the robot for example rebooted and the LED light was still on.