Einführung in die Mikrocontroller-Programmierung

Gruppe D

Steckbrief

- Julia Kisela
  Physik (B. Sc.)

- Nicolai Kaschta
  Informatik (B. Sc.)

Gliederung

1. Aufgabenstellung
    - Herangehensweise
    - Praktikumsverlauf
2. Ergebnisse
    - Mikrocontroller
    - Lustige Bugs
    - Theorie zum Farbsensor
    - Code Farbsensor
3. Ausblick und Reflektion

Aufgabenstellung

- Grundlagen der Mikrocontroller-Programmierung kennen lernen

- Kleine Projekte mit dem Atmega

Praktikumsverlauf

- Zusammenbau: Verzögerung wegen fehlender Bauteile
- Kein kontinuierliches Arbeiten, eher Hackathon-Style
- Programmierung relativ straightforward

Microcontroller allgemein

- Vereinen mehr Komponenten als Prozessoren
- System on a Chip

- Für Regel und Steueraufgaben in denen Schaltungen nicht hinreichend sind

- Kostengünstig

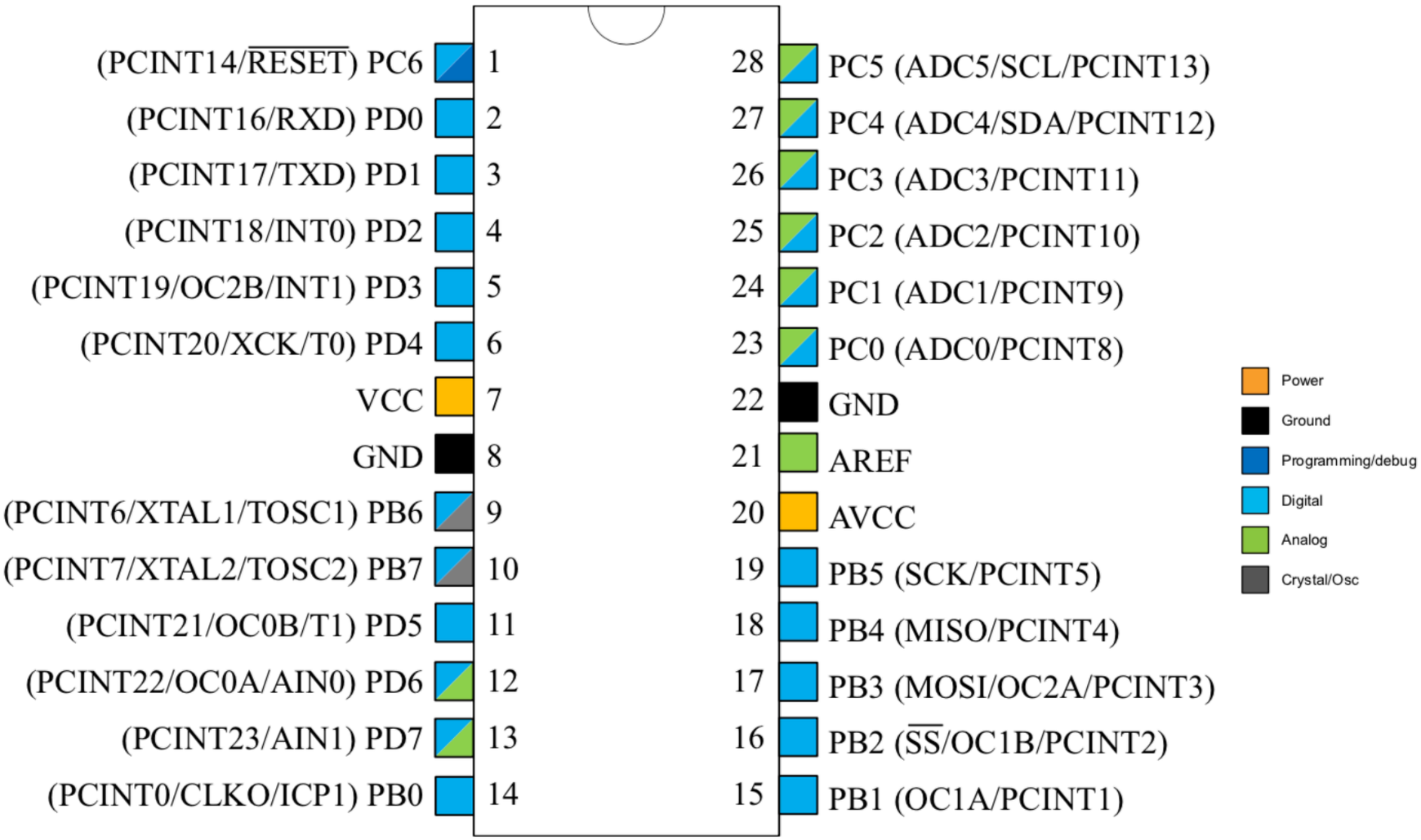
Microcontroller Atmega168

- 8-Bit Prozessor mit max. 20MHz
- 1 kb SRAM-Datenspeicher
- 16 kb Flash-Programmspeicher
- 512 Bytes EEPROM

Microcontroller Atmega168

- Eingebauter A/D Wandler
- Timer für die Pulsweitenmodulation
- Mehrere Pins zur Ein- und Ausgabe
- Serielle Schnittstelle (UART)

Microcontroller Atmega168



Microcontroller Platine

- Spannungsregler
- Power-LED
- 16 MHz-Quarz
- 10-polige ISP-Buchse
- Reset-Taster
- Zwei PMOS-Transistoren

Hardwareprobleme

- Spannungsversorgung
- Kontaktprobleme des Microcontrollers (gemessen)
- Scara-Roboter verhielt sich unvorhergesehen
- Defekte Bauteile (vor allem LEDs)

Softwareprobleme

- Kalibrierung aufwändig
- Unterscheidung Rot-Magenta physikalisch schwierig
- Einfluß von Umgebungsfaktoren schwer abzuschätzen

Lustiger Bug

- Falsch:

```
value16 = getADCValue(ADC_KANAL);
// schneide 2 Bits vom 10-Bit-ADC-Wert (des Potis) ab,
// um einen 8-Bit-Wert zu bekommen
value = (uint8_t) value16>>2;
```

- Richtig:

```
value16 = getADCValue(ADC_KANAL);
// schneide 2 Bits vom 10-Bit-ADC-Wert (des Potis) ab,
// um einen 8-Bit-Wert zu bekommen
value = (uint8_t) (value16>>2);
```

Lustiger Bug

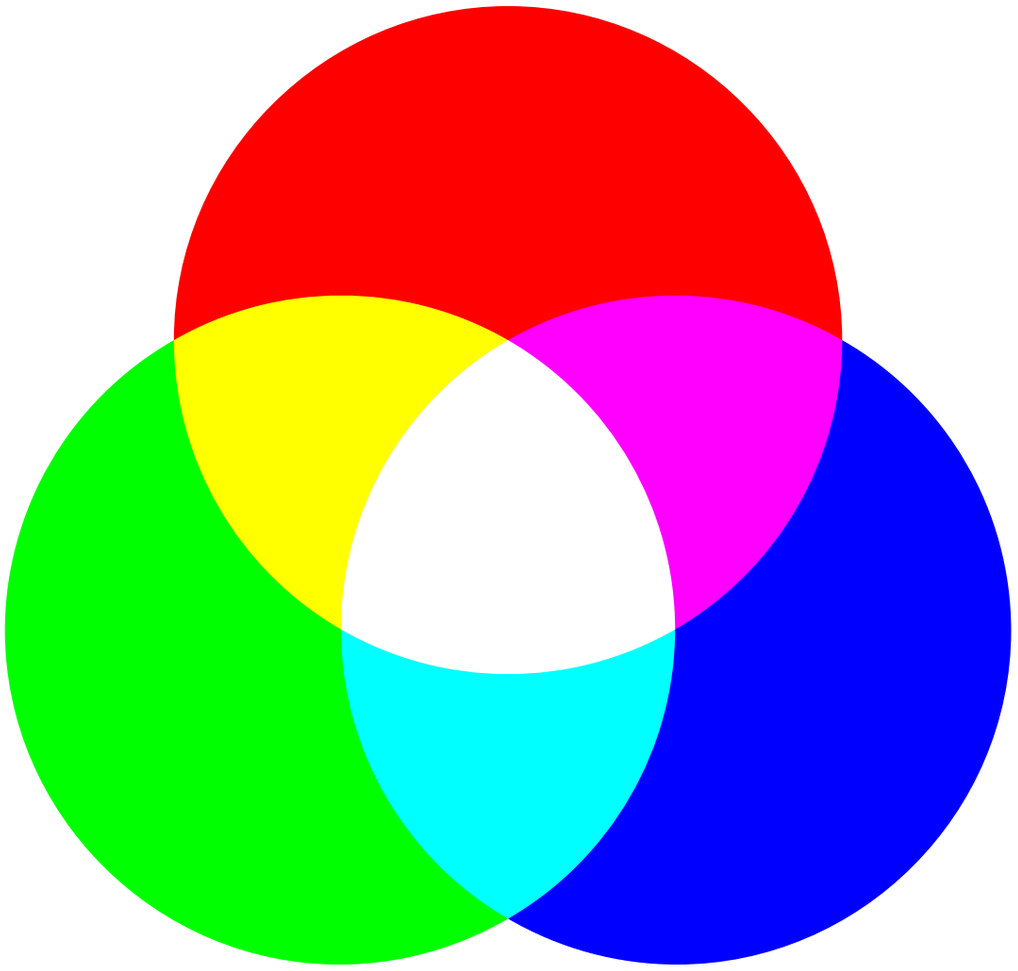
```
value = (uint8_t) value16>>2;

value = 0000 00AB CDEF GHIJ

(uint8_t) value = CDEF GHIJ

            >>2 = 00CD EFGH ~~IJ~~
```

```
value = (uint8_t) (value16>>2);

value      = 0000 00AB CDEF GHIJ

value >>2 = 0000 0000 ABCD EFGH ~~IJ~~

(uint8_t) =           ABCD EFGH
```

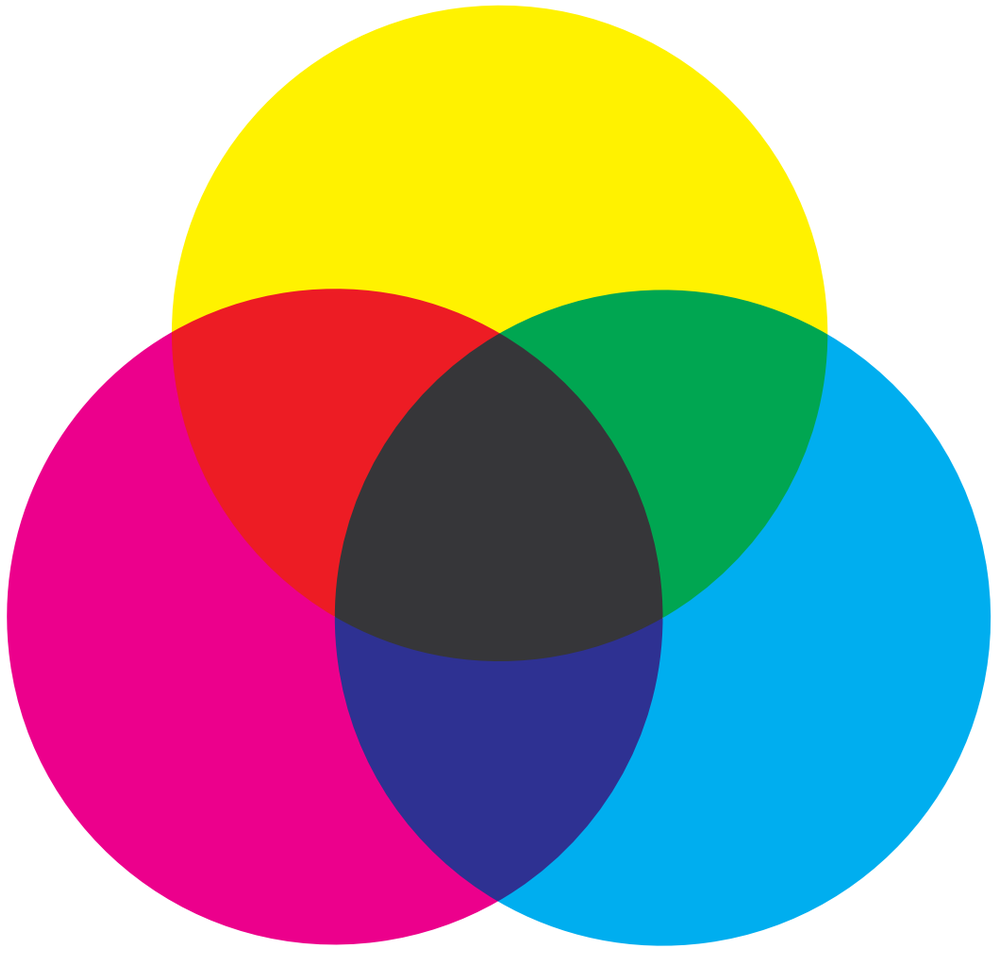
Farbsensor

- Gehäuse schützt Meßobjekt vor Fremdlicht
- Komponenten: RGB-LED und Phototransistor
- Controller leuchtet abwechselnd mit R, G, B und mißt die einfallende Helligkeit
    - Farben unterscheiden?

Farbmischung



Additiv (RGB)



Subtraktiv (CMY)

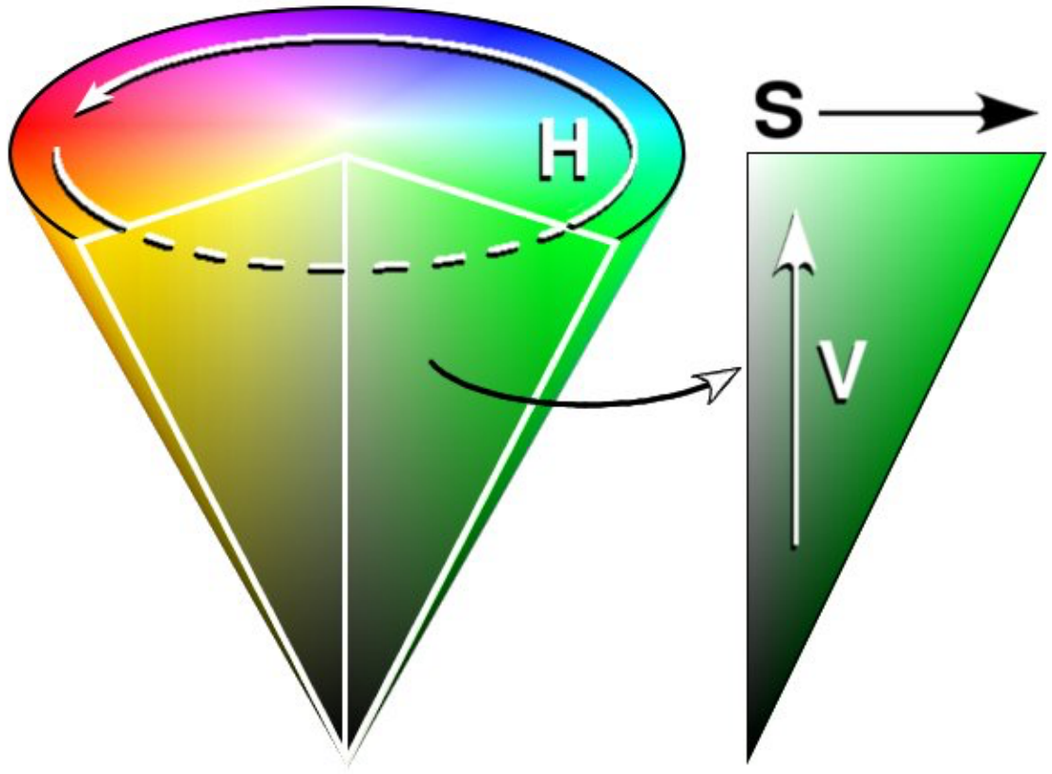
HSV-Farbmodell

- **H**ue, **S**aturation, **V**alue

- **Farbton**
  0° ≙ Rot
  120° ≙ Grün
  240° ≙ Blau

- **Sättigung**
  0% ≙ Grau/Weiß/Schwarz
  100% ≙ knallbunt

- **Helligkeit**
  0% ≙ Schwarz,
  100% ≙ volle Helligkeit



HSV-Farbmodell

$$MAX := \max(R, G, B),\ MIN := \min(R, G, B)$$

$$H := \begin{cases} 0, & \text{falls } MAX = MIN \Leftrightarrow R = G = B \\ 60° \cdot \left(0 + \frac{G-B}{MAX-MIN}\right), & \text{falls } MAX = R \\ 60° \cdot \left(2 + \frac{B-R}{MAX-MIN}\right), & \text{falls } MAX = G \\ 60° \cdot \left(4 + \frac{R-G}{MAX-MIN}\right), & \text{falls } MAX = B \end{cases}$$

$$\text{falls } H < 0° \text{ dann } H := H + 360°$$

$$S := \begin{cases} 0, & \text{falls } MAX = 0 \Leftrightarrow R = G = B = 0 \\ \frac{MAX-MIN}{MAX}, & \text{sonst} \end{cases}$$

$$V := MAX$$

HSV-Farbmodell

- Vorteile
- Logische Trennung der Eigenschaften der Farbe
- Farbton im Spektrum

Code (main)

```c
// Pins für R, G, B-LEDs
#define R_PIN (1<<0)
#define G_PIN (1<<1)
#define B_PIN (1<<2)

// Array für die drei Pins (um Code nicht duplizieren zu müssen)
uint8_t pins[] = {R_PIN, G_PIN, B_PIN};
#define N_PINS 3
```

```c
int main(void) {
    uint8_t i; // Zählvariable
    init(); // Initialisierungen

    for (i = 0; i < N_PINS; i++) {
        DDRC |= pins[i]; // Schalte alle Pins auf Output
    }

    while (1) {
        measure(); // Miß jede Sekunde und gib die Farbe aus
        _delay_ms(1000);
    }
}
```

```
// Sensorkanal
#define SENSOR 5

// Dunkelspannungen
#define R_DARK 0
#define G_DARK 0
#define B_DARK 0

// Maximalwerte des ADC für maximale Helligkeit
#define R_NORM 100.0
#define G_NORM 100.0
#define B_NORM 180.0

// Array für die drei Pins (um Code nicht duplizieren zu müssen)
uint8_t pins[] = {R_PIN, G_PIN, B_PIN};
#define N_PINS 3

void measure() {
    uint16_t values[N_PINS]; // Array für ADC-Werte für jede Farbe
    uint8_t i; // Zählvariable

    for (i = 0; i < N_PINS; i++) { // Für jede Farbe...
        PORTC |= pins[i]; // Mach entsprechende LED an
        _delay_ms(100); // Warte kurz, damit der Readout stabil wird
        values[i] = getADCValue(SENSOR); // Lies den Phototransistor aus
        PORTC &= ~pins[i]; // LED aus
    }

    float r = (values[0] - R_DARK) / R_NORM; // Rotwert auf 0..1 normieren
    float g = (values[1] - G_DARK) / G_NORM; // Grünwert auf 0..1 normieren
    float b = (values[2] - B_DARK) / B_NORM; // Blauwert auf 0..1 normieren

    float h, s, v; // Diese Variablen werden gleich die HSV-Werte erhalten

    rgb2hsv(r, g, b, &h, &s, &v); // Konvertiere RGB-Meßwerte zu HSV (s. o.)

    raw_output(r, g, b, h, s, v); // Gib nochmal alle Werte aus (macht Kalibrieren einfacher)

    decide(h, s, v); // Entscheide Dich für eine Farbe
}
```

```c
// Konvertiere rohe RGB-Werte nach HSV
// Argumente h, s, v sind Zeiger, damit ihre Werte
// an den Aufrufer zurückgegeben werden können
void rgb2hsv(float r, float g, float b, float *h, float *s, float *v) {
    float max = MAX(MAX(r, g), b); // Größter der drei Werte R, G, B
    float min = MIN(MIN(r, g), b); // Kleinster der drei Werte R, G, B

    if (max == min) { // Größter = Kleinster => alle gleich
        *h = 0; // Hier ist der Hue-Wert beliebig
    } else if (max == r) { // Rotwert ist der größte
        // Wähle 0° (Rot) als Basis und addiere Grün minus Blau
        *h = 60 * (0 + (g-b)/(max-min));
    } else if (max == g) { // Grünwert ist der größte
        // Wähle 120° (Grün) als Basis und addiere Blau minus Rot
        *h = 60 * (2 + (b-r)/(max-min));
    } else if (max == b) { // Blauwert ist der größte
        // Wähle 240° (Blau) als Basis und addiere Rot minus Grün
        *h = 60 * (4 + (r-g)/(max-min));
    }

    // Normierung auf 0..360°
    while (*h < 0) { *h += 360; }

    // Wenn max = 0, wollen wir nicht durch max dividieren...
    if (max < 0.0001) {
        // ...sondern setzen die Sättigung direkt auf Null
        *s = 0.0;
    } else { // Sonst...
        // ...ist die Sättigung das Verhältnis von der
        // Spannweite der Werte (Max-Min) zum größten
        *s = (max-min)/max;
    }

    *v = max; // Der Helligkeitswert ist der größte Meßwert
}
```

```c
// Maximum von a und b
#define MAX(a, b) ((a>b)?(a):(b))
// Minimum von a und b
#define MIN(a, b) ((a<b)?(a):(b))
```

```c
void decide(float h, float s, float v) {
    if (v < 0.2) { // Wenn Helligkeitswert klein (kann kalibriert werden)
        uart_puts("Black\r\n"); // schwarz
        return; // für diese Messung nichts weiter ausgeben
    }
    if (v > 0.8 && s < 0.2) { // Wenn Helligkeit hoch, aber Sättigung klein
        uart_puts("White\r\n"); // weiß
        return; // nichts weiter ausgeben
    }

    // Wir teilen den Farbkreis (Hue) im Folgenden in sechs Stücke
    // für Rot, Gelb, Grün, Cyan, Blau, Magenta

    // unkalibriert:
    //if (h <= 30.0 || 330.0 < h) { // 330°..360° und 0°..30°
    // kalibriert:
    if (h <= 30.0) { // 0°..30°
        uart_puts("Red\r\n"); // Rot
    } else if (30.0 < h && h <=90.0) { // 30°..90°
        uart_puts("Yellow\r\n"); // Gelb
    } else if (90.0 < h && h <= 150.0) { // 90°..150°
        uart_puts("Green\r\n"); // Grün
    } else if (150.0 < h && h <= 210.0) { // 150°..210°
        uart_puts("Cyan\r\n"); // Cyan
    } else if (210.0 < h && h <= 270.0) { // 210°..270°
        uart_puts("Blue\r\n"); // Blau
    // unkalibriert:
    //} else if (270.0 < h && h <= 330.0) { // 270°..330°
    // kalibriert:
    } else if (270.0 < h) { // 270°..360°
        uart_puts("Magenta\r\n"); // Magenta
    }
}
```

Ausblick

- Veränderung: Drei Phototransistoren mit Farbfiltern
  - Auch aktive Farbquellen messen
- Praktikum allgemein: Gute Grundlage für zukünftige Mikrocontroller-Projekte
  - Jetzt können wir uns an speziellere Projekte wagen :)

Danke für Eure Aufmerksamkeit!