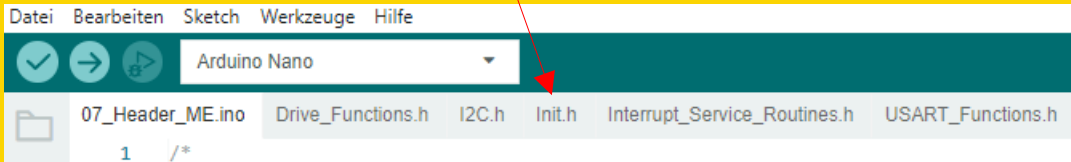


Montage und Programmierung  
eines Roboters für  
**ROBOCUP JUNIOR RESCUE**  
mit Arduino Nano  
Teil 2.5: Header

Jetzt machen wir einen seeeehr großen Schritt!

Wir räumen das Programm noch mehr auf und wir binden noch zusätzliche „Libraries“ ein.

Wir lagern alles, was im „setup“ passiert in einen separaten Header aus. Den nennen wir „Init.h“. (Initialisierung)



Den Header könnt ihr euch gerne genauer ansehen, indem ihr auf den Reiter klickt. (Nur wenn ihr tiefere Einblicke gewinnen wollt!)

```
7  #ifndef INIT_H_
8  #define INIT_H_
9
10 void Ping_LCD(void);
11 void Ping_BNO055(void);
12 void BNO055_Init(void);
13
14 unsigned char LCD_ON_OFF; //0 -> Off, 1 -> On
15 unsigned char BNO055_ON_OFF; //0 -> Off, 1 -> On
16
17 //Initialize the Microcontroller
18 void Init (void) {
19     //Port settings
20     //0 = Input, 1 = Output
21     DDRB = 0b00111111; //00n.n,1LED,1Servo,1Motor_B,1PWM_B,1PWM_A,1MotorA
22     DDRC = 0b00000000; //00n.n,0SCL,0SDA,1Push_1,111Dist-4 to -2
23     DDRD = 0b10000000; //1US_Trig,0Push_2,0BLnWH,0Colour,0Encoder,0US_Echo,0TX,0RX
24     PORTD = 0b00001000; //Pull Up Resistor for encoder
25     PORTC = 0b00001000; //Pull Up Resistor for Push_1
26     //Interrupt settings
27     //External
28     EICRA = 0b00001010; //INT0, INT1, falling edge at RD0(US) and 1
29     EIMSK = 0b00000011; //Enable INT0, enable INT1
30     //Internal
```

Dort werden zusätzlich Funktionen und Variablen deklariert.

Statt jedem Pin einzeln die Datenrichtung zuzuweisen [pinMode(pin, In/Out)] legen wir sie direkt in die entsprechenden Register ab, was ja **pinMode** auch macht. (DDR steht für Data Direction Register)

0 = Input, 1 = Output

So wie in der Pindescription erwähnt stellen wir bei PORTD den Pin 7 auf Output und den Rest auf Input.

Statt **pinMode(pin, INPUT\_PULLUP)** schreiben wir **PORTx = 0b00100001**, wenn wir an einem Port Pin 0 und Pin 5 als PullUp benutzen wollen.

Danach folgt die Einrichtung von externen Interrupts, z.B. für den Ultraschall, fallende Flanke, an Pin 2 und 3 und schalten sie ein.

(Nicht abgebildet:

Timer, Analoge und I2C Einstellungen)

Außer den Headern die wir schon kennen, **Drive\_Funktions.h** und **Init.h**, schauen wir uns auf der nächsten Seite noch ein wenig die anderen Interrupts an.

Als erstes ist da der **TIMER2\_Overflow Interrupt**:

Alle **128us** wird das laufende Programm durch das „Überfließen“ des Timer 2 ein Interrupt ausgelöst.

Overflow meint hierbei, wenn das TCNT2 (Timer/Counter2)

Register von 255 auf 0 springt. Das geschieht alle 128us.

(Der Prescaler des TMR2 hat dabei den Wert 8. Register TCCR2B Bits 0 bis 2.)

Mit Hilfe dieser „Uhr“ steuern wir

- das Blinken der **LED**
- die Erfassung der **Analog**werte
- die Erfassung des **Farb**sensorwertes
- die Initialisierung des **Ultraschall**sensors
- die Übermittlung der Sensorwerte an den **Computer**
- die Übermittlung der Sensorwerte an das **LCD**
- die Erfassung des **Gyro**-Sensors

Diese Prozesse laufen alle **nebenher**. „Parallel“ zum Hauptprogramm. Dort müssen wir die Werte „nur“ noch verarbeiten.

Außerdem, wie schon besprochen, erfassen wir über **INT0** den **US-Wert** und außerdem über **INT1** den **Encoder**-Wert.

Dann haben wir noch den Header

**USART-Functions.h**.

Er beinhaltet die Funktionen, die wir benötigen, um die Werte an den Computer zu senden.

Zum Schluß ist da noch

**I2C.h**.

Mit diesem Header steuern wir die Kommunikation mit dem Gyro und dem LCD.

Der Data\_Visualizer sieht jetzt anders aus:

Wir müssen nicht mehr jedes Datum (Singular von Daten) einzeln mit **Serial.print** auf den Weg bringen. Wir müssen den Daten, die wir anzeigen wollen, nur eine Stelle anweisen, an der sie erscheinen sollen.

**Data[0]** wird auf dem Arduino-Monitor ganz **links** angezeigt, **Data[9]** ganz **rechts**.

Auf dem LCD werden nur die ersten 8 Werte angezeigt, **Data[0] bis [3]** in der **oberen** Zeile, **Data[4] bis [7]** in der **unteren**.

**Lasst euch feste Werte an unterschiedlichen Stellen anzeigen, um ein Gefühl für die Anzeige zu bekommen.**

# Wozu das alles?

Bisher haben wir im Hauptprogramm dafür gesorgt, dass die Sensorwerte **erfasst und angezeigt** werden.

Das ist mühsam und wir müssen immer dafür sorgen, dass das zu jeder Zeit erfolgt: Also müssen wir jeden Programmteil überprüfen, ob wir wirklich alles bekommen, was wir brauchen!!!

Das umgehen wir, indem wir im **INTERRUPT** alles **automatisch** in bestimmten Zeitabständen generieren.

Ihr müsst also nur noch im Data\_Visualizer festlegen, was ihr, an welcher Stelle angezeigt bekommen wollt.

Wenn ihr an der **ersten Stelle** des Monitors einen bestimmten Wert anzeigen lassen wollt, müsst ihr die Zuweisung zu **Data[0]** ändern.

z.B.: **Data[0] = Abstand\_rechts\_hinten;**

Sonst müsst ihr nichts ändern.

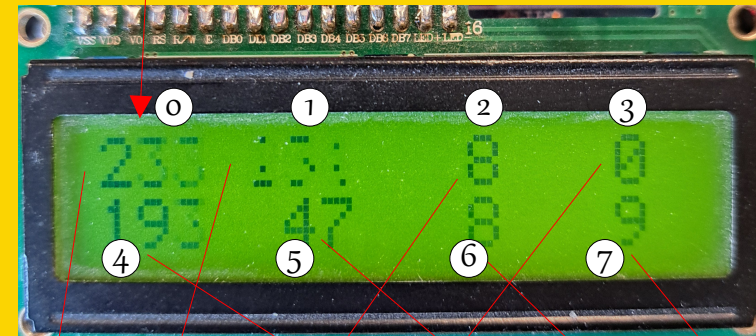
Das Programm im **INTERRUPT** sorgt dafür, dass Abstand\_rechts\_hinten an der entsprechenden Stelle angezeigt wird.

Beachtet, dass ihr auf dem LCD nur die ersten 8 Werte, also Data[0] bis Data[7] anzeigen lassen könnt!

```
//Data[0] bis Data[7] werden auf LCD angezeigt
//Data[8] und Data[9] nur auf dem Monitor
void Data_Visualizer (void) {
  Data[0] = z++;
  Data[1] = Abstand_rechts_vorne;
  Data[2] = US_Time;
  Data[3] = Black_n_White;
  Data[4] = Colour; //HI-Byte
  Data[5] = Encoder; //LO-Byte
  Data[6] = Start_Taster;
  Data[7] = Gyro_X;

  //Nur auf Data Visualizer angezeigt
  Data[8] = Gyro_Y;
  Data[9] = Gyro_Z;
}
```

- Variable zählt hoch
- IR-Sensor vorne
- Ultraschall
- Schwarz/weiß-Sensor
- Farbsensor
- Zähler Radumdrehung
- Taster hinten
- Drehwinkel um die Senkrechte



Auf dem Monitor werden Data[0] bis Data[9] angezeigt.

131	130	8	0	192	47	8	9	1	0
131	130	8	0	192	47	8	9	1	0
132	243	8	0	192	47	8	9	1	0
133	219	8	0	193	47	8	9	1	0
133	191	8	0	193	47	8	9	1	0

Data 0 1 2 3 4 5 6 7 8 9

Montage und Programmierung  
eines Roboters für  
**ROBOCUP JUNIOR RESCUE**  
mit Arduino Nano  
Teil 2.6: ?