# Was ist der Arduino

Ein Microcontrollerboard (um 20 Euro oder um 3 in China) mit einer einfachen IDE (integrated devolopment environment) zum Programmieren in C bzw C++. Als IDE eignet sich aber auch VisualStudioCode mit der PlatformIO Erweiterung sehr gut, die ca 500 Controller , wie zB den ESP32 unterstützt.

Der Arduino ist ein Mikrocontroller, dh er hat kein Betriebssystem, multitasking oder multithreading kennt er nicht, man muss sich selber darum kümmern und unabhängige Programmteile in kleine Zeitscheiben zerlegen, wie es sonst der scheduler macht. Dann kann man so was ähnliches bauen, muss sich aber selber darum kümmern, das die Teile nacheinander drankommen und sich nicht gegenseitig ausbremsen oder aufhalten.

Der Arduino besitzt mehrere digitale Ein und Ausgänge und ein paar analoge Eingänge, deren Werte von einem 10 Bit AD-Wandler in 1024 Stufen zwischen 0 und 5V dargestellt werden.  
Analoge Ausgänge lassen sich nur über Pulsweitenmodulation erzeugen:

* ein ständiger Wert am Ausgang erzeugt 5V
* wenn der Wert immer kurz gleichlang ein und ausgeschaltet wird 2,5V
* sonst eben im Verhältnis der Ein und Ausschaltdauer

Das fertig compilierte Programm wird dann auf den Arduino übertragen und rennt dort sobald die Übertragung fertig ist.

Deshalb besteht ein Arduinoprogramm immer aus zwei Funktionen:

setup() … da werden Startwerte festgelegt

loop() … wie der Name schon sagt eine Endlosschleife mit der Funktionalität, das wäre wie main und darin ein while(true)

Programmiert wird in C++, dh man kann auch Objekte erzeugen, Methoden verwenden usw  
Viele Steuerungsprogramme kommen aber mit funktional programmierten C aus.

Der Arduino ist ein single task   
  
wichtige Befehle am Arduiono

digitalWrite(pin,spannung); setzt die Leitung auf 0 (LOW) oder 5V (HIGH) bzw

x=digitalRead(pin); schaut ob der pin HIGH oder LOW ist

x=analogRead(pin); liest die Spannung am pin ein. Von 0V (x ist 0) bis 5V (x ist 1024)

delay(ms); wartet ms lang

x=millis(); Zeitmessung, Millisekunden, die seit dem Start des Arduino vergangen sind

# Arduino lesen vom und schreiben auf den seriellen Port (USB)

Serielle (USB) Schnittstellenbefehle:

Die serielle Schnittstelle ist ein FIFO Buffer, wenn er voll ist, wird der älteste Wert gelöscht.  
Werte sind normalerweise durch Zeilenumbrüche getrennt.  
Lesen vom Buffer holt sich den ältesten Eintrag und löscht ihn aus dem Buffer, die anderen rücken nach

serial.begin() da legt man zB die Übertragungsrate fest

serial.available() schauen ob etwas im seriellen buffer des Arduino drinnen steht

x= serial.read, y=serial.readbyte holts aus dem eigenen Buffer

serial.print(byte), serial.println(String), schreibt in den Buffer der Gegenstelle

# Zusammenarbeit Arduino Windows, Sprachsteuerung, Motor schalten

## Amazon Alexa als Sprachsteuerung verwenden

Man muss ein Konto bei AWS (Amazon Web Services) anlegen. Man legt dann skills an, das sind Sprachbefehle die Alexa erkennt. Amazon stellt für die Kommunikation mit Alexa für Java und Node.JS Frameworks zur Verfügung. Dabei kann man alexa auch rein als mächtige Spracherkennungssoftware nutzen.  
Ablauf wäre so:  
Benutzer spricht intent aus (zB alexa, schalte motor ein), der gesprochene Satz wird an amazon gesendet, dort analysiert, in einen string umgewandelt und als response bekommt man ein json Objekt zurück, das den string enthält zB „motor ein“. Das json Objekt auswerten und den dazu passenden Steuerbefehl dann zB über die serielle Schnittstelle an den Arduino weitersenden.

## Google Cloud Speech-to-Text

Ist ein Service von google mit eigener api, man nimmt das gesprochene auf, es wird zu google gesendet, dort mit neuralen Netzen analyssiert un man bekommt als response den Text als string zurückgesendet. Ist gratis für eine gewisse Anzahl an Minuten, die nalysiert werden (60 min) danach zahlt man nach Zeit. Vorteil: sehr gute Erkennung, Rauschunterdrückung, schnell  
Nachteil: man braucht permanente Verbindung zur cloud

Dh man schreibt einen eventhandler, der durch das Mikrofon ausgelöst wird, schickts in die cloud und wertet den string aus

## Windows Sprachsteuerung

Sprachsteuerung ist in Windows integriert (ab Win 7 funktioniert sie gut, Probleme falls deutsches Windows und englische Visual Studio), alles läuft lokal ab.

Lernen des Frequenzmusters des Benutzers (arbeitet lokal, nicht wie alexa und co), deshalb muss man die Spracherkennung trainieren, in dem man ihr etwas vorliest

Danach erkennt Windows wenn ein gesprochenes Wort der Beschriftung eines Buttons der GUI im Vordergrund entspricht und emuliert das Drücken des Buttons, dh der event-Handler onclick wird ausgelöst

Wenige Worte, die sich von der Aussprache deutlich unterscheiden, möglichst einsilbig (ein, aus, stop, los) - nur für Motor geht es aber gut, auch wenns laut ist im Hintergrund.

Gutes Mikrofon mit Rauschunterdrückung.

Windows GUI mit Buttons auf denen die Schlüsselwörter stehen

LOS

STOP

* + Im Event Handler des Buttons, OnButtonClick 🡪 USB Schnittstelle ansprechen
  + USB ansprechen (siehe auch Beispielcode im Anhang)

geht in C++ über die CSerial-Klasse, die in Visual Studio dabei ist oder über die System.IO.Ports;

dann legt man eine Instanz des des Objekts SerialPort an   
beim Konstruktor gibt man die Portnummer (zB COM2) an danach verwendet man die Methoden open, read, write, close (ähnlich zu file IO)

SerialPort mySerialPort = new SerialPort("COM2");  
 mySerialPort.RtsEnable=true;

mySerialPort.open();

mySerialPort.write(“…..“)

usw

das RtsEnable ist eigentlich eine alte Handshake Leitung der seriellen Schnittstelle (ready to send),   
funktioniert stabiler mit diesem Befehl ,obwohl er beim USB keine Rolle spielt, die IO.Ports baut das aber nach.

Will man Daten (zb die eines Sensors) empfangen, baut man eine asynchrone callback Routine zum DataReceived Event

mySerialPort.DataReceived += new SerialDataReceivedEventHandler(HoleDaten);

* + am Arduino
    - SerialRead => Digital Pin ein oder Ausschalten (DigitalWrite) in Schleife

240 V

Relais

PC

Arduino

USB

USB

GND

0 5V

init{}  
loop()  
{

if Serial.available())

{

X=Serial.readString();

Und dann x auswerten

}

delay(100); //polling dh jede Sekunde nachschauen ob was gekommen ist

}

Man muss sich ein eigenes Protokoll überlegen zB einen passenden String, der die nötigen Steuerinformationen enthält

Um stärkere Verbraucher zu schalten benötigt man Relais, dieses schaltet mit einer kleinen Spannung eine Große.

### Überlegungen zur Echtzeitfähigkeit

* Arduino arbeitet in der Schleife, ist echzeitfähig, single Task Prozessor
* Arduino kann keine echten asynchronen Prozesse sondern eigentlich nur polling (es gibt noch Interrupts) dh man muss in regelmäßigen Abständen nachschauen ob was gekommen ist.
* USB ist Streitfall ob echtzeitfähig, zum isochronen Transfer zwingen (der ist echtzeitfähig)
* Windows NEIN (Uhr ist zwar sehr genau, Windows schaut aber nur alle ca 40ms auf der „Uhr“ nach wie spät es ist, kleinere Zeiten werden nicht aufgelöst. Man weiß dann zwar, dass und um wieviel es zu spät ist, aber das darf bei echtzeitfähigen Systemen nicht passieren)

# Arduino mit GSM/LTE shield zum Versenden von SMS - Aufgabe Remote Temperaturanzeige

## Lösungen

* Hochwertigere Heizungssteuerungen bieten die Möglichkeit der Remote-Kontrolle, benötigen dazu aber eine permanente Internetverbindung und üblicherweise die Einrichtung einer vpn Verbindung oder einen dyndns Service, damit der Netzwerk von außen erreichbar bleibt, wenn sich die IP-Adresse ändert.
* Es gibt um 80 Euro fertige Module die über einen SIM-Kartenslot verfügen und bei Unterschreiten einer gewissen Temperatur eine SMS senden.

Selber programmieren:

* Ein PC mit einem AD/Wandler und Internetstick. Einfach zu programmieren aber teuer.
* Ein Raspberry Pi mit AD Wandler und USB Internet Stick (kommt auf ca 70 Euro)
* Selber bauen mit einem Arduino, einen einfachen Temperatursensor und einem GSM-Shield. Hardware-Kosten Arduino Klon aus China 5 Euro, Temperatursensor 2 Euro, GSM Shield 30 Euro, Netzteil 5 Euro, ergibt < 50 Euro.
* Teurer wird’s mit LTE shields für den Arduino, die kosten ab 50 Euro, sind aber zuverlässiger

## Umsetzung:

Arduinos besitzen digitale Ein- und Ausgänge und Analoge Eingänge. Ein digitaler Ausgang wird auf 5V gesetzt, wenn er gesetzt wird. Das kann als Spannungsversorgung für einen Sensor verwendet werden. Die Spannung am Temperatursensor würde dann je nach Temperatur zwischen 0V und der Versorgungsspannung 5 V liegen.  
Die Analogeingänge des Arduino wandeln eine Spannung in eine Zahl um. Bei  
5 V wird eine 1023 erzeugt bei 0 V eine 0. So bekommt man zur aktuellen Temperatur einen Messwert zwischen 0 und 1024

## Temperaturerfassung

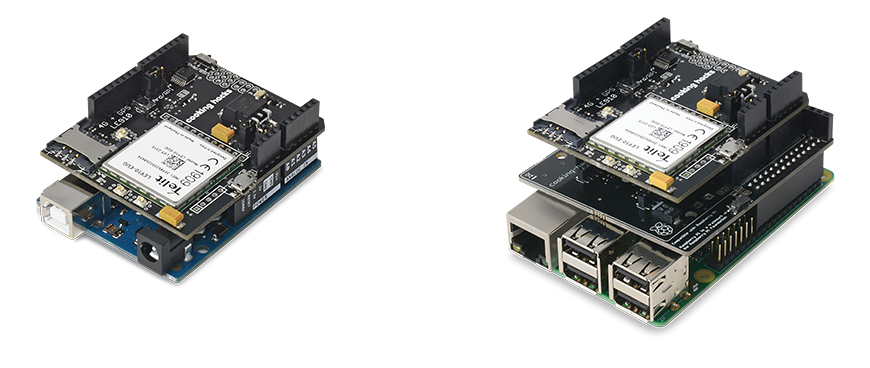
Für die Messung reicht ein einfacher PTC – Widerstandssensor, den man bei jedem Elektronik Händler um unter 2 Euro bekommt (PTC- positiver Temperatur Koeffizient bedeutet, dass je höher die Temperatur desto höher der Widerstand und desto höher die Spannung ist).  
Zum Kalibrieren liest man den Analogwert bei zwei verschiedenen Temperaturen ein und stellt zwei Gleichungen auf. Da der Zusammenhang linear ist gilt  
T1 = k \* Analogwert1 + d  
T2 = k \* Analogwert2 + d

Daraus kann man k und d ermitteln und erhält eine Formel für die Umrechnung der analogwert in Temperaturen

temp = (k \* analogRead(tempPin) + d)

## Kommunikation Arduino – GSM-Modem

Das Senden der SMS erfolgt über ein sogenanntes GSM shield, das einfach auf den Arduino aufgesteckt wird. Shields sind kleine Platinen deren Anschlüsse so angebracht sind, dass sie übereinander gestapelt auf das Arduinoboard aufgesteckt werden können und dessen Funktionalität auf diese Weise erweitern.



Die Kommunikation erfolgt über den I2C bus oder eine serielle Schnittstelle zwischen Arduino und shield. Der Arduino sendet über die serielle Schnittstelle AT Befehle, die die Funktion des shields steuern

Dieser AT-Befehlssatz dient zur Steuerung eines Modems. Für die Versendung von SMS werden zB folgende Befehle benötigt:

* AT+CMGF=1
* AT+CMGS="+436642753600"

Der Erste AT-Befehl signalisiert dem Chip, dass eine SMS verschickt werden soll. „AT+CMGS“ startet den SMS Prozess mit der gegebenen Telefonnummer. Nach diesem Befehl signalisiert der Chip mit „>“ dass eine Nachricht eingegeben werden kann. Anschließend wird die Nachricht mit Strg + Z oder dem Hexcode „0x1a“ bestätigt werden.

Neben der Möglichkeit Kurznachrichten zu senden kann man übrigens auch Anrufe tätigen. Der Audio Out- und Input erfolgt über die Klinkenbuchsen des Shields.

Eventuell wäre für den Arduino noch eine kleine unabhängige Spannungsversorgung (eine Powerbank) praktisch, die noch solange Strom liefert, dass der Arduino noch schnell eine SMS senden kann, wenn zB der Strom ausfällt. Auch das lässt sich leicht messen wenn zB ein Stromdruchflusszähler um das Stromkabel, das zum Arduino führt, gelegt wird und dieser Sensor ebenfalls an eine Analogeingang angeschlossen wird. Bei Stromausfall würde die Spannung auf 0 fallen.

## Beispielcode (Pseudocode) am Arduino

ein Arduino Programm hat immer die Funktionen setup zum Initialisieren und loop zum Ausführen der Aufgabe

void setup() //das ist die Standardroutine die immer am Beginn ausgeführt wird

{

k und d setzen  
Grenztemperatur setzen, unterhalb der eine SMS gesendet werden soll  
serielle Schnittstellenparameter zum shield setzen  
digitalen Port auf true setzen, der den Sensor mit Strom versorgt

}

void loop() //das wird ununterbrochen ausgeführt  
{

T= k \* AnalogRead(Sensorpin) + d  
 if(T<Grenzwert und gesendet==false)  
 {

AT Befehle an das shield senden um SMS zu starten  
Warten bis > zurückkommt  
SMS Text an das shield senden  
0x1a senden, damit sms abgeschickt wird  
gesendet auf true setzen, damit nicht alle 5 Sekunden eine SMS gesendet wird

}

delay(3600); /jede Minute eine Messung machen

}

## BeispielCode für das Einlesen von Werten vom Arduino in C# und Anzeigen als floating chart in einem canvas

Das Programm liest Werte von der USB Schnittstelle ein, sobal sie ankommen. Im canvas wird ein array von 500 Werten angezeigt. Wenn ein neur Wert dazukommt werden alle vorhandene Werte nach links verschoben (dh. Der älteste fällt weg) und der neue Wert wird hinten angehängt.

Das canvas kann nur polylines anzeigen, deshalb wird jeder Messwert einem Punktobjekt zugeordnet, die Punkte einer Punkt-Collection (dürfte eine Liste sein) und die Punkt-Collection der polyline und diese dann im canvas als Child hinzugefügt, dh gezeichnet

Man muss mit dem dispatcher arbeiten, weil der eventhandler keinen Zugriff auf die Controls der GUI hat.

using System;

using System.Collections.Generic;

using System.IO;

**using System.IO.Ports;**

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows;

using System.Windows.Controls;

using System.Windows.Data;

using System.Windows.Documents;

using System.Windows.Input;

using System.Windows.Media;

using System.Windows.Media.Imaging;

using System.Windows.Navigation;

using System.Windows.Shapes;

**using System.Windows.Threading;**

namespace Wert\_vom\_Arduino\_AG1

{

/// <summary>

/// Interaktionslogik für MainWindow.xaml

/// </summary>

public partial class MainWindow : Window

{

SerialPort mySerialPort;

int[] mw= new int [500]; //array zum Aufnehmen der Messwerte

public MainWindow()

{

InitializeComponent();

mySerialPort = new SerialPort("COM10");

for (int i = 0; i < 500; i++) mw[i] = 25;

}

private void Button\_Click(object sender, RoutedEventArgs e)

{

if (!mySerialPort.IsOpen)

{

mySerialPort.Open();

mySerialPort.RtsEnable = true;

//Wenn Daten im seriellen Buffer sind, geh in den event Handler

mySerialPort.DataReceived += new SerialDataReceivedEventHandler(HoleDaten);

}

}

private void HoleDaten(object sender, SerialDataReceivedEventArgs e)

{

//Daten sind angekommen - lies sie aus und schreib sie in die Textbox

SerialPort sp = (SerialPort)sender;

try

{

String indata = sp.ReadExisting();

Application.Current.Dispatcher.Invoke(DispatcherPriority.Normal, (Action)delegate () { Wert\_vom\_USB.Text = indata; });

//array der Messwerte nach links verschieben und neuen Wert als letzten Wert dazuhängen

int x= Convert.ToInt32(indata);

for (int i = 0; i < 499; i++) mw[i] = mw[i+1];

if(x>=0 && x<=50) mw[499] = x;

Application.Current.Dispatcher.Invoke(DispatcherPriority.Normal, (Action)delegate () { zeichnen(); });

}

catch

{

}

}

private void zeichnen()

{

//MessageBox.Show("zeichnen...");

Point Punkt = new Point();

Polyline pg = new Polyline();

pg.Stroke = System.Windows.Media.Brushes.Black;

pg.StrokeThickness = 1;

PointCollection Punkte = new PointCollection();

for (int i = 0; i < 500; i++) //Punktcollection neu aufbauen

{

Punkt.X = i;

Punkt.Y = 100-mw[i];

Punkte.Add(Punkt);

}

int x = mw[499];

pg.Points = Punkte;

mycanvas.Children.Clear();

mycanvas.Children.Add(pg);

}

private void Button\_Click\_1(object sender, RoutedEventArgs e)

{

if (mySerialPort.IsOpen) mySerialPort.Close();

}

}

}