Sensor Daten Qualität

Oft hat man es mit nicht idealen Sensorwerten/Daten zu tun. Rauschen oder zu geringe Auflösung erschweren das Arbeiten mit den Sensorwerten. Um dies zu verbessern gibt es viele Möglichkeiten. (Filtern, Mehrfach-Sampeln, Offset rausrechnen, anpassen der analogen Messreferenz, Verstärken des Signales, etc...).

Um eine gute Steuerung aus Sensordaten zu realisieren ist es wichtig die Charakteristik der Sensordaten zu Verstehen. Oft kann man schon bei mit einfacher Visualisierung per (Serial)Plot erkennen was man tun kann um bessere Datenqualität für seine Anwendung zu erhalten.

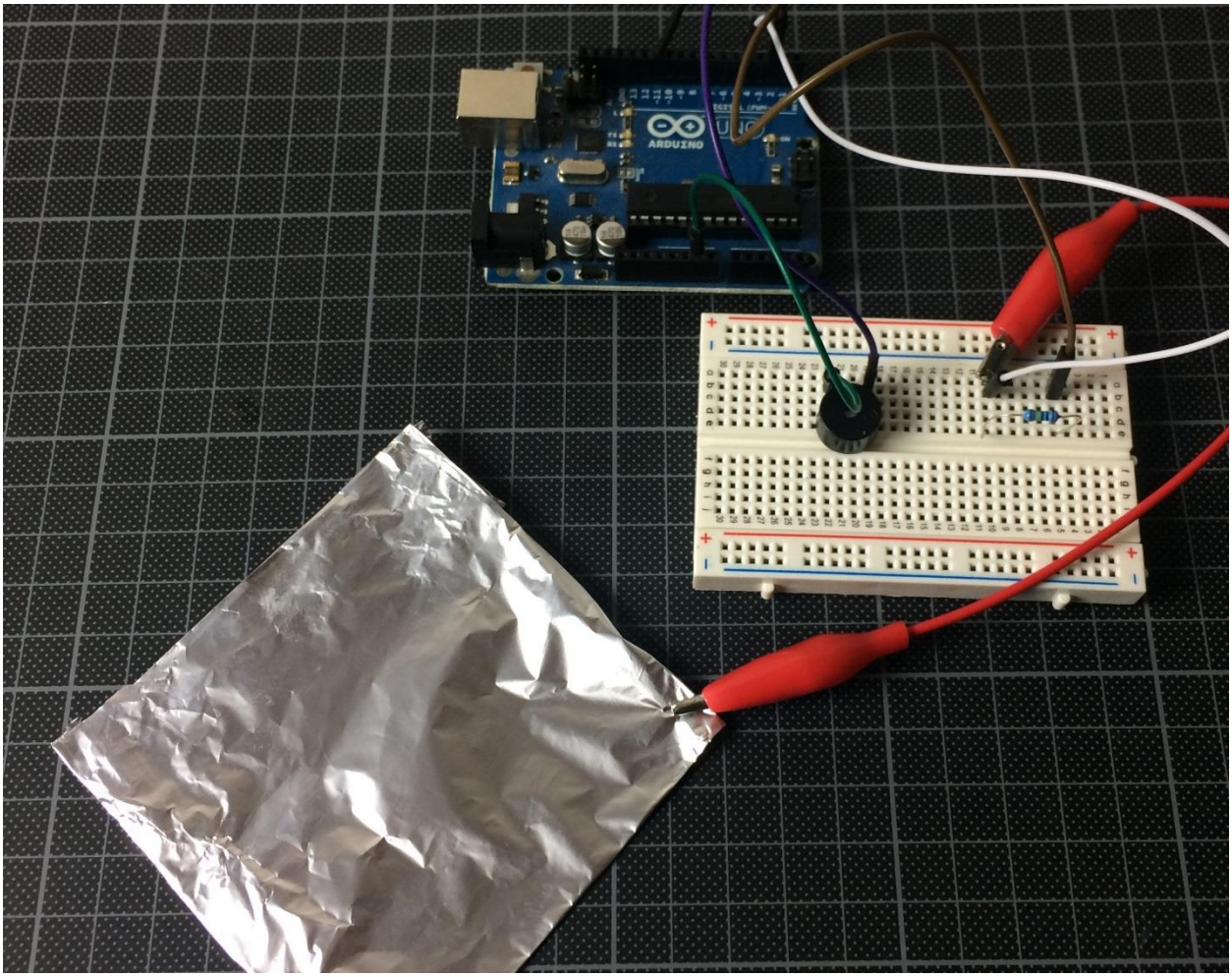


Abbildung 1 Arduino/Steckbrett/Kapazitiver-Sensor

Beschreibung der Übung:

Wir wollen Daten eines Kapazitiven Sensors lesen und als Ton mit einem Beeper ausgeben.

Ein Kapazitiver-Sensor ist schnell und einfach gebaut aber seiner Sensordaten richtig und robust zu nützen ist dagegen etwas aufwendiger.

Man kann mit etwas Alu-Folie und einem (hohen) Widerstand einen Sensor bauen.

Praktischer weise gibt es schon eine Arduino Library für Kapazitive Sensoren.

http://playground.arduino.cc/Main/CapacitiveSensor

Library downloaden und im Arduino libraries Verzeichnis entpacken.

Alternativ kann man eine Library auch als Zip in der Arduino IDE hinzufügen.

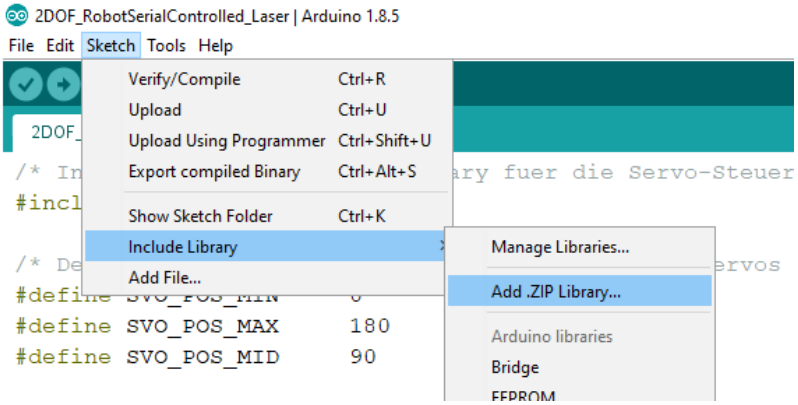


Abbildung 2 Add Library from Zip File

Verkabelung:

Stecke den Wiederstand auf das Steckbrett und verbinde den Wiederstand mit Pin 4 und Pin 2.

Klemme das Kroko-Kable mit einem Ende an den Wiederstand an Pin und das andere Ende an die Alu-Folie (bzw .Blech oder Metalstueck).

Komponente	**Verbindung**
1. Wiederstand 1 – 10 MegaOhm zwischen Pin 4 und 2	Arduino Pin 4 -> (Steckbrett – Wiederstand) -> Arduino Pin 2
2. Krokokabel mit Alu-Folie	Arduino Pin 2 -> Krokokabel mit Alu-Folie
	Arduino Ground Pin -> Ground (blau) am Steckbrett
(*Extra Uebung) Beeper	Aruino Pin 11 -> Beeper + Arduino Ground -> Beeper -

*Extra Uebung Beeper

Mit diesen Hilfs Funktionen kann man einfach Töne mit einem Beeper abspielen.

```
int toneBuffer[] = {
440,466,494,523,554,587,622,659,698,740,784,831,880,932,988,1047,1109,1175,1245
,1319,1397,1480,1568,1661,1760,1865,1976,2093,2217,2349,2489,2637,2794,2960,0 };

void PlayToneFromMittelWert(long mittelWert)
{
	if (mittelWert < 30) UpdateBeeper(-1);
	else
	{
		int toneNr = map(mittelWert, 30, 100, 0, 20);
		UpdateBeeper(toneNr);
	}
}

void UpdateBeeper(int ton)
{
	if (ton <0 || ton > 32) noTone(11);
	else tone(11, toneBuffer[ton]);
}
```

Übung 1: Lesen des Kapazitiven-Sensors und Ausgeben

```
////////////////////////////////////////////////////////////////////////////////////
#include <CapacitiveSensor.h>

CapacitiveSensor   cs_4_2 = CapacitiveSensor(4, 2);  // 10M resistor between pins 4 & 2,
// pin 2 is sensor pin, add a wire and or foil if desired

void setup()
{
	cs_4_2.set_CS_AutocaL_Millis(0xFFFFFFFF);
	Serial.begin(115200);
}

void loop()
{
	long value = cs_4_2.capacitiveSensor(30);
	Serial.println(value);
}
////////////////////////////////////////////////////////////////////////////////////
```

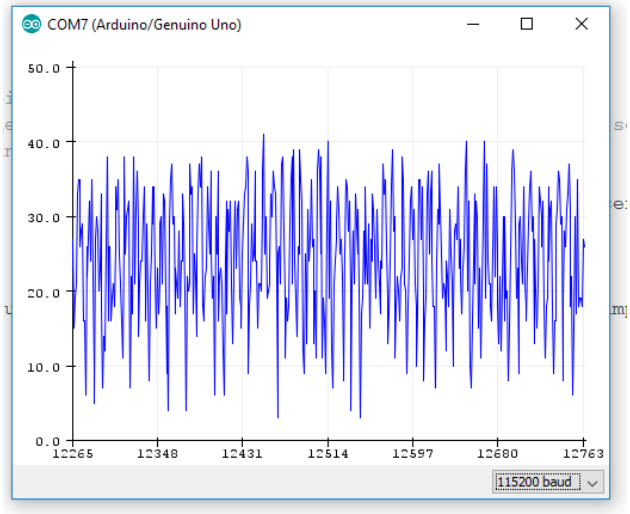


Abbildung 3 Kapazitiver Sernsor Werte (Idle)

Man sieht schnell warum das Signal einen „wild“ verrauscheten Charakter hat und aber auch sehr hohe Werte annehmen kann.

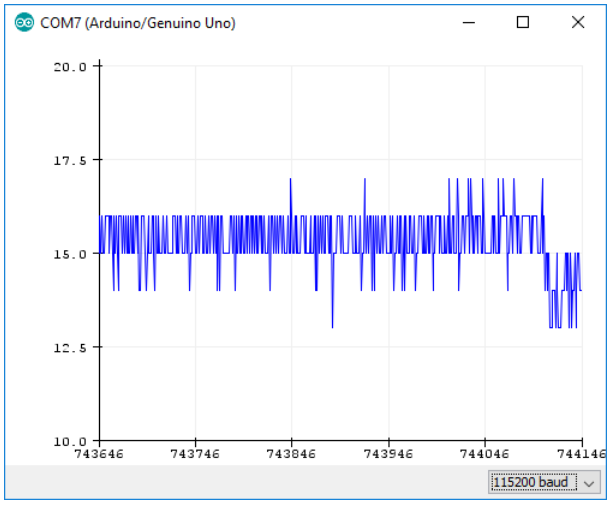
Aber man sollte auch bemerken das man die Werte mit der Hand schon ein paar Zentimeter entfernt beeinflussen kann.

Übung 2: Mehrfach Lesen (Oversampling)

Durch bilden einen Durchschnitt-Wertes von mehreren Samples kann man das Signal schon um einiges verbessern.

Schreibe eine Funktion mit der du den Mittelwert aus 16 Messungen bildest.

```
////////////////////////////////////////////////////////////////////////////////////////////////

#include <CapacitiveSensor.h>

CapacitiveSensor   cs_4_2 = CapacitiveSensor(4, 2);

void setup()
{
	cs_4_2.set_CS_AutocaL_Millis(0xFFFFFFFF);
	Serial.begin(115200);
}

void loop()
{
	// 4-fach Sampling
	long value1 = cs_4_2.capacitiveSensor(30);
	long value2 = cs_4_2.capacitiveSensor(30);
	long value3 = cs_4_2.capacitiveSensor(30);
	long value4 = cs_4_2.capacitiveSensor(30);

	long value = (value1 + value2 + value3 + value4) / 4;

	Serial.println(value);
}

////////////////////////////////////////////////////////////////////////////////////////////////
```



Man merkt dass das Rauschen schon weniger wird aber dass das Messen länger dauert.

Übung 3: Mehrfach Lesen und Buffer (Buffered-Oversampling)

Durch speichern der Signalwerte in einem Buffer und bilden des Mittelwertes der im Buffergespeichert Werte kann die anzahle des Messungen reduziert werden. Dies verkürzt die Dauer der Messungen.

Erzeuge einen Buffer für 64 Werte und befülle in Schrittweise mit jeweils 4 Messwerten in einem Zyklus.

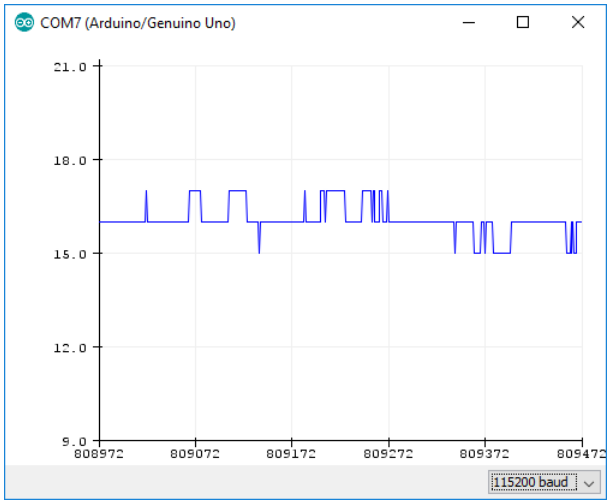


Abbildung 4 Mehrfach-Sampling mit Buffer Groesse 64

Erweitere dein Programm um diese Funktionen und benütz sie für deine Ausgabe.

```
#define BUFFER_SIZE    64
int index = 0;
long messBuffer[BUFFER_SIZE];

void BefuelleBuffer(int anzahl)
{
	for (int i = 0; i < anzahl; i++)
	{
		messBuffer[index] = cs_4_2.capacitiveSensor(30);
		index = (index + 1) % BUFFER_SIZE;
	}
}

long MittelWertBuffer(int BitErweitern)
{
	long summe = 0;
	for (int i = 0; i < BUFFER_SIZE; i++)
	{
		summe += messBuffer[i];
	}
	return summe / (BUFFER_SIZE - BitErweitern);
}
```

Übung 4a: Einfacher Filter

Man kann ein Verrauschtes Signal Glätten in den letzten Messwert in den aktuellen Wert hingerechnet.

zB: `valueLast = (valueLast * 2 + valNow) / 3; // Hierbei wird die Vergangenheit doppelt so stark gewichtet wie der aktuelle Messwert.`

```
long letzterWert = 0;

void loop()
{
	long value = cs_4_2.capacitiveSensor(30);

	// Filtering
	letzterWert = (float)(value + letzterWert * 4) / 6.0f;

	Serial.println(letzterWert);
}
```

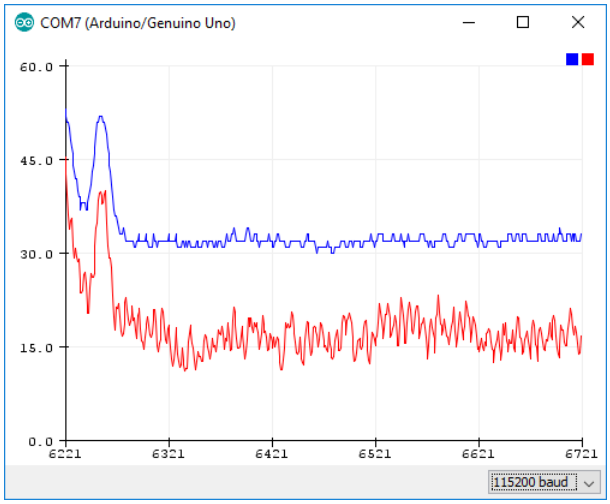


Abbildung 5 Rote Line ist gefiltertes Signal - Blau ist Oversampling

Übung 4b: Einfacher Filter

Erhöhen sie den Vergangenheitsanteil auf 40 und Vergleichen sie die Signale.

```
	letzterWert = (float)(value + letzterWert * 40) / 41.0f;
```

Übung 4c: Vergleichen sie Oversampling und Filter Signale

Plotte alle 3 Signale (Oversampling, Filter 1/4 und filter 1/40).
Spiele ein bisschen mit den Filter und Oversampling Parameter und Plotte die Signale.

Übung 5: Steuer des 2DOF Roboters mit zwei Kapazitiven Sensoren

Schließe den 2DOF Roboter an den Arduino an. (siehe Verkabelung aus letzte Lehreinheit)

Schließe einen weiteren Kapazitiven Sensor an.

Modifiziere dein Programm so dass du einen Servo zwischen 0 und 180 Grad bewegst.
Durch einen Sensor soll der Winkel Wert kleiner werden und mit dem zweiten Sensor soll der Winkelwert grösser werden.

Folgende Hilfsfunktion soll dir dabei helfen.

```
#define MIN_VAL     40
#define MAX_VAL     100

float servoPos = 0.0f;

float ChangeServo(int sensorVal1, int sensorVal2)
{
	if (sensorVal1 > MAX_VAL) sensorVal1 = MAX_VAL;
	if (sensorVal2 > MAX_VAL) sensorVal2 = MAX_VAL;

	if (sensorVal1 > MIN_VAL)
	{
		servoPos -= (sensorVal1 - MIN_VAL) / (MAX_VAL - MIN_VAL) / 2;
	}

	if (sensorVal2 > MIN_VAL)
	{
		servoPos += (sensorVal2 - MIN_VAL) / (MAX_VAL - MIN_VAL) / 2;
	}

	if (servoPos < 0) servoPos = 0;
	if (servoPos > 180) servoPos = 180;

	return servoPos;
}
```