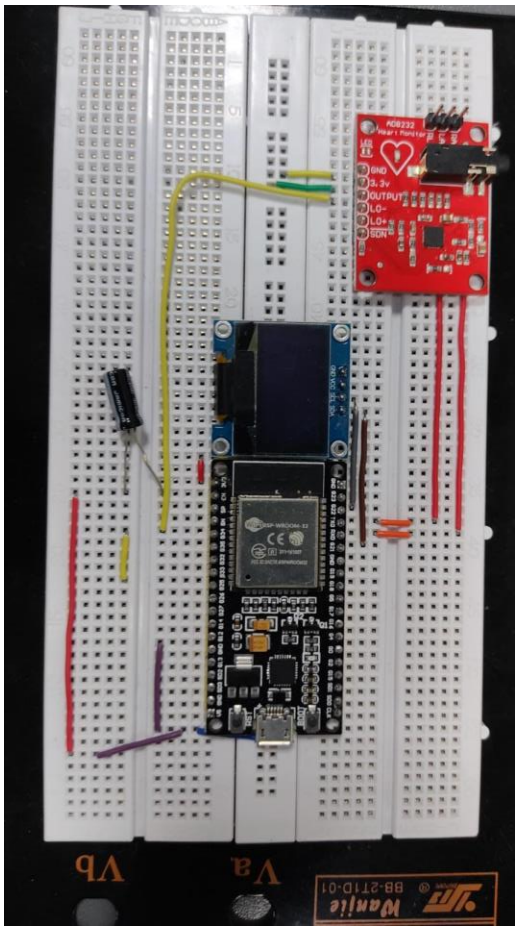
Hochschule Karlsruhe
University of
Applied Sciences

Fakultät für
Maschinenbau und
Mechatronik



EKG-Laborbericht

Im Sommersemester 2023

EKG-Daten Aufnahme und Verarbeitung

Christian Sánchez (sach1024)
Fernando Gallardo (gafe1012)

MECB721 – Ausgewählt Kapitel der Mechatronik

1. Inhalt

2. Vorwort zur Aufgabenstellung

Im Bereich der Fach Ausgewählt Kapitel der Mechatronik wird eine Aufgabe vorgestellt, welche die Erfassung und Verarbeitung von EKG-Daten mittels eines EKG-Recorders und des Mikrocontrollers ESP32-WROOM-32 beinhaltet. Zur Umsetzung dieses Vorhabens wurde ein EKG-Gerät bestehend aus einem Mikrocontroller, einem OLED-Display sowie einem Pulssensor zur Aufnahme der EKG-Signale konstruiert. Der ESP32-WROOM-32 Mikrocontroller ist in der Lage, die Analogsignale des AD8232-Pulssensors zu verarbeiten und die Daten per Jumper-Kabel an das Display weiterzuleiten. Des Weiteren werden die erfassten EKG-Daten per WLAN und UDP-Protokoll an einen Computer gesendet, um in der Datenanalyse-Software Matlab weiterverarbeitet und analysiert zu werden.

Als Entwicklungsumgebung für die Umsetzung dieses Vorhabens wurde die Programmiersprache C++ in Verbindung mit Visual Studio Code und der PlatformIO-Erweiterung eingesetzt. Die Grundlage für die erfolgreiche Durchführung dieser Aufgabe bildeten die im Rahmen der Vorlesung Informationstechnik erworbenen Kenntnisse im Bereich der Signalverarbeitung und Filterung von Signalen.

Das Forschungsprojekt wurde von einem 2er-Team in einem Zeitraum von sechs Wochen bearbeitet. Hierbei wurden wöchentlich Arbeitsblätter verwendet, um die Funktionsweise der einzelnen Komponenten des EKG-Geräts zu verstehen und sich mit ihnen vertraut zu machen. Im Rahmen des fortlaufenden Berichts werden neben den erreichten Erfolgen auch die im Verlauf des Projekts aufgetretenen Schwierigkeiten thematisiert

3. Einführung in EKG

Die Elektrokardiographie (EKG) ist eine etablierte Untersuchungsmethode zur Darstellung der Herzfunktion. Hierbei werden die elektrischen Spannungen, welche durch die Aktivität der Herzmuskelfasern generiert werden, gemessen und in Form einer Herzspannungskurve graphisch dargestellt (s. Abb. 2-1). Durch eine Analyse dieser Kurve können Informationen über den Gesundheitszustand des Herzens gewonnen werden. Zur Durchführung der Untersuchung werden Elektroden, welche mit einem leitfähigen Gel versehen sind, auf die Haut des Patienten geklebt, um die Strom- bzw. Spannungsschwankungen an verschiedenen Stellen der Haut zu messen. Im Rahmen der vorliegenden Laboruntersuchung werden insgesamt drei Messelektroden verwendet, wovon zwei in der Nähe der Brust und eine auf Höhe der Rippen platziert werden (s. Abb. 2-1).

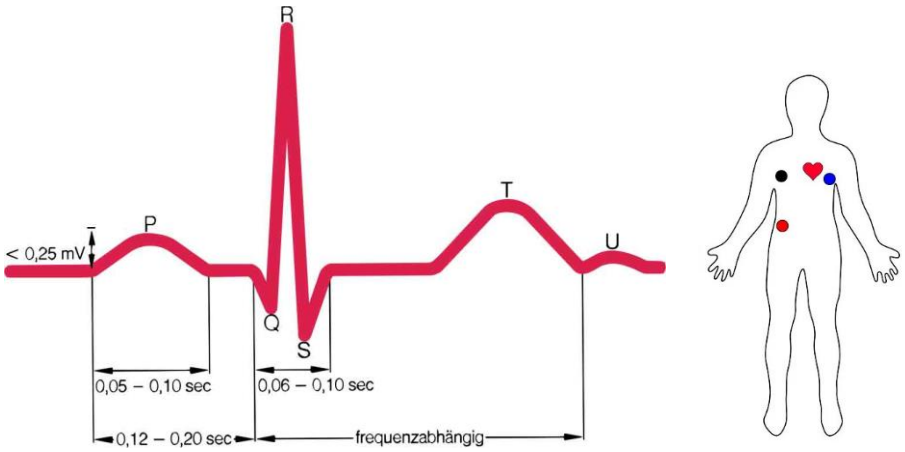


Abbildung 3-1 EKG- Kurve (links) und Lage der Elektroden (rechts)

Der QRS-Komplex hat bei erwachsenen Patienten üblicherweise eine Dauer von 0,06 bis 0,10 Sekunden. Eine Dauer von über 0,12 Sekunden wird als abnormal betrachtet. Um alle Zacken des QRS-Komplexes erfassen zu können, sollte die Abtastfrequenz mindestens das Doppelte der höchsten im QRS-Signal vorkommenden Frequenz betragen (50 Hz), um dem Nyquist-Shannon-Abtasttheorem zu entsprechen (Miron, 2020) Im Rahmen der vorliegenden Laboruntersuchung wurde daher eine Abtastfrequenz von 250 Hz gewählt, was einem Abtastintervall von 4 ms entspricht.

4. Hardwarebeschreibung

Im Rahmen des Informationstechnik-Labors wurde ein Versuchsaufbau zur Messung der Herzfunktion mit folgenden Hardware-Komponenten realisiert:

- einer Entwicklungsplatine NodeMCU mit ESP32-WROOM-32
- einem 1,3 Zoll OLED I2C 128 x 64 Pixel Display
- einem EKG-Shield mit AD8232 Modul und drei Messelektroden.

ESP

Der ESP32-WROOM-32 ist ein 32-Bit-Mikrocontroller mit integrierter Wi-Fi-Schnittstelle, der 12 I/O-Pins, sowie I²C-, I²S-, SPI- und UART-Schnittstellen bietet (EWALD, 2023). Der Analog-Digital-Umsetzer an Pin A0 ermöglicht die Umwandlung von analogen Signalen im Bereich von 0 bis 3,3 Volt in Integer-Zahlen zwischen 0 und 1024. Die Kommunikation zwischen dem MCU und OLED-Display erfolgt über das I²C-Protokoll, welches eine serielle Datenübertragung ermöglicht und bis zu 128 Teilnehmer miteinander kommunizieren lässt. Die Stromzufuhr für die Module wird über die entsprechenden Pins gewährleistet.

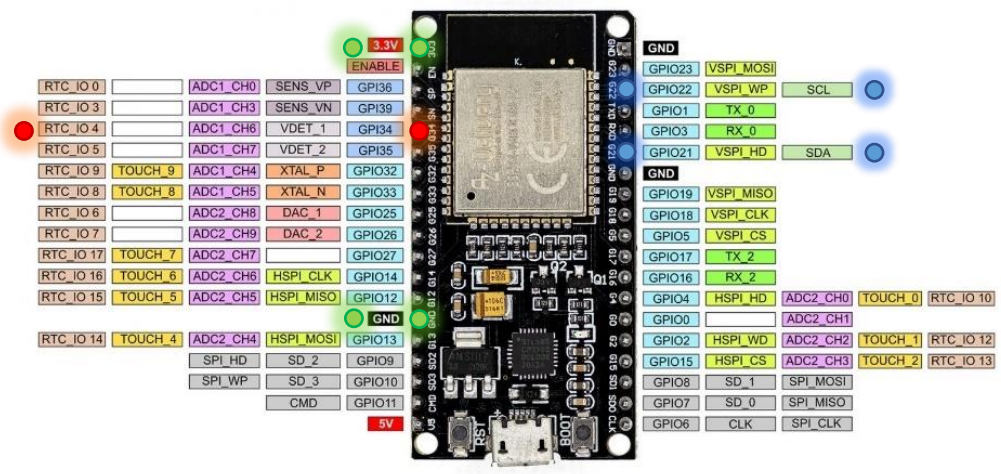


Abbildung 4-1 ESP32-WROOM-32Pinout (EWALD, 2023)

In Abbildung 3-1 sind alle Ein- und Ausgänge des ESP dargestellt, wobei die Pins, die für das Projekt genutzt wurden, farblich markiert sind. Die blauen hinterlegten Ports dienen zentral zur Datenübertragung zwischen der weiteren Hardware und dem ESP. Der analoge Eingang, über den die EKG-Daten an den Prozessor geleitet werden, befindet sich oben links roten Punkt in der Abbildung.

OLED DISPLAY

Das Display dient neben dem seriellen Monitor am PC als Informationsanzeige bezüglich WiFi Status, Matlab-Verbindung und EKG-Daten. Das Display besitzt eine Auflösung von 64 Pixel in der Höhe und 128 Pixel in der Breite und benötigt für die vollständige Inbetriebnahme vier Kabel, davon zwei für die Stromversorgung und zwei für das Versenden von Daten an das Display (s. Abb. 4-2).

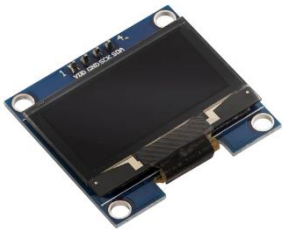


Abbildung 4-2 ESP32 OLED Display

Die Datenübertragung erfolgt unidirektional, da das Display nur Daten anzeigen muss und keine generieren wird. Neben verschiedenen Einstellungsmöglichkeiten seitens der Software ist es in erster Linie wichtig, dass Text, Zahlen und einzelne Pixel, die daraufhin mit Linien verbunden werden, angezeigt werden können (AZ-Delivery). Das EKG-Signal wird hierbei dann nach erfolgreicher Inbetriebnahme und Konfiguration kontinuierlich wiedergegeben.

Da es viele verschiede Ausführungen und daher auch Ansteuerungen des Displays gibt, müssen vorgefertigte Bibliotheken eingebunden werden, um die Bedienung des Displays zu vereinfachen. In diesem Projekt wurde hierfür die Bibliothek ***„ESP8266 and ESP32 OLED driver for SSD1306 displays“*** verwendet. Die Wahl dieser Bibliothek wurde aufgrund ihrer Kompatibilität mit dem ESP-Mikrocontroller und der Anzeige getroffen. Durch die Verwendung dieser Bibliothek wurde die Implementierung und Verwendung des Displays vereinfacht und beschleunigt.

EKG

Das EKG-Modul AD8232 ist die dritte und letzte Hardwarekomponente, die für die Signalaufzeichnung und Weiterleitung sorgt. Wie bereits erwähnt, ist sie sowohl mit Jumper-Kabeln, die auch hier wieder zwecks Stromversorgung und Datenübertragung erforderlich sind, mit dem ESP32-WROOM-32 als auch über drei Elektroden mit einem Menschen verbunden (s. Abb. 4-3).

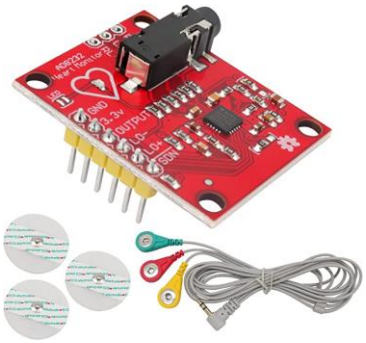


Abbildung 4-3 EKG-Model AD8232 mit Jumper--Kabeln und Elektroden

Der Sensor liefert Daten zwischen 1 und 1024, welche an den analogen Eingang des Mikrocontrollers übertragen werden. Dabei handelt es sich um ungefilterte Daten, die im späteren Verlauf noch näher analysiert werden. Je nach Programm können hier verschiedene Abtastfrequenzen eingestellt werden, je nachdem wie viele Daten pro Sekunde benötigt werden. Die versendeten Daten werden dann im ESP32-WROOM-32 gespeichert und nach Verarbeitung an das Display sowie an Matlab geleitet. (Analog Devices)

Laboraufbau

Gemäß der Anleitung wurden alle Hardwarekomponenten auf dem Steckbrett miteinander verbunden, wie in den Abbildung 4-4 dargestellt. Auf der linken Seite sind die allgemeinen Anschlüsse, die in der Vorlesung des Ausgewählten Kapitels der Mechatronik gezeigt werden und auf der linken Seite die tatsächliche Anpassung und Verbindung zum ESP32-WROOM-32, der in diesem Projekt verwendet wurde. Der nächste Schritt besteht darin, die Entwicklungsumgebung auf dem PC einzurichten. Eine zusätzliche Verbindung, die sich für das Testen neuer Codes als praktisch erwies, war die Verwendung eines 100uF-Kondensators, der mit der PIN EN verbunden ist. Dadurch wird verhindert, dass wir bei jeder Code-Aktualisierung die RST-Taste drücken müssen.

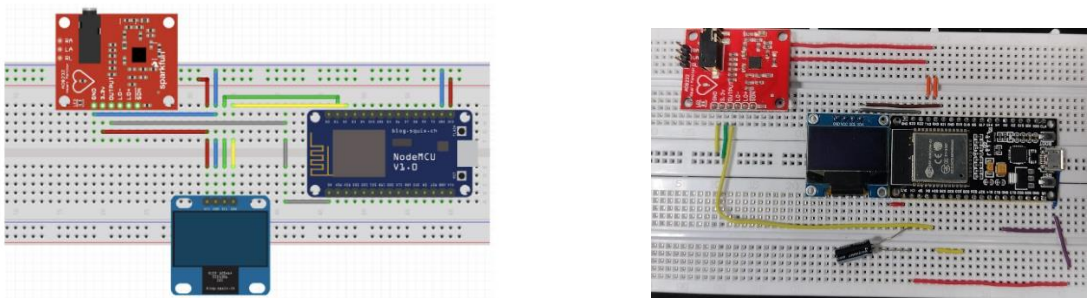


Abbildung 4-4 Verdrahtung der Hardware-Skizze (links) und Verdrahtung Umsetzung(rechts)

5. Einrichtung der Entwicklungsumgebung

5.1 Einführung in der Einrichtung der Entwicklungsumgebung

Zu Beginn des Projekts wurden die notwendigen Programme aktualisiert und installiert:

- Matlab Version R2022b
- Python 3.10
- Platform IO
- Visual Studio Code

Zuerst muss MatLab über WLAN-Verbindung ein Datenpaket an VScode senden (s. Abb. 5-1). In dieser Information werden die IP und UDP-Port übermittelt, die benötigt werden, um die Daten vom Mikrocontroller an Matlab zu senden.

```
% UDP Verbindung setzten
    u = udpport;
    RemoteIPLab = "192.168.188.197";
    write(u,"Hi ESP, hier ist Matlab","char",RemoteIPLab,4210);
```

Abbildung 5-1 Matlab Code: UDP-Verbindung

Um die Codeverständlichkeit und Wartbarkeit zu erleichtern, wurde das Projekt in verschiedene Dateien aufgeteilt: display.h, einlesen.h, WLAN_connection.h und main.c. Zusätzlich zu diesen Dateien müssen die Bibliotheken „ESP8266TimerInterrupt“ (von Khoi Hoang) und „ESP8266 and ESP32 OLED driver for SSD1306 displays“ (von ThingPulse) in das Projekt installiert werden. Im Folgenden wird der Code anhand der Dateiorganisation erklärt.

5.2 WLAN_connection.h

Diese Datei basiert sich auf dem Befehl WiFi.begin(), wobei das erste Argument der Name des WiFi-Netzwerks und das zweite das Passwort ist.

```
WiFi.begin("FRITZ!Box 7590 VL", "56616967766283031728");   // Connect to the lab network
```

Abbildung 5-2 VS-Code: WLAN-Verbindung

Um sicherzustellen, dass nach 30 Sekunden eine Fehlermeldung angezeigt wird, wird ein Zähler eingestellt, der erhöht wird, solange der Befehl WiFi.status(), der anzeigt, ob eine Verbindung hergestellt wurde oder nicht, nicht WL_CONNECTED zurückgibt. Wenn die Verbindung hergestellt ist, wird die lokale IP und der UDP-Port auf dem Bildschirm angezeigt.

```
while (WiFi.status() != WL_CONNECTED) { // Wait for the Wi-Fi to connect
delay(1000);
Serial.print(++i);
Serial.print(' ');
if ( i >= TimeOUT) {   // Timeout, wenn Verbindung nicht hergestellt werden kann
  Serial.print("Connection to Wifi-Network failed for 30 seconds ");
  i = 0;
  }
}
```

Abbildung 5-3 VS-Code: Wenn nach 30 Sekunden noch keine Verbindung hergestellt wurde, erscheint eine Fehlermeldung.

5.3 Datenaufnahme (einlesen.h)

Es enthält die Funktion, die alle 4 Millisekunden Messungen von den Elektroden nimmt. Dazu wird das Interrupt des Mikrocontrollers verwendet (der in setup() konfiguriert ist, s. Abb. 5-9). Mit der Funktion analogRead() werden die Daten vom angegebenen Port (in unserem Fall Port 34) erfasst und in der Variablen data_RAW gespeichert. Dann werden sie im 15000-Positionen-Array *bufferRawAndFiltered* und im 7500-Positionen-Array *bufferFilteredForDisplay* gespeichert (s. Abb. 5-4).

```
data_RAW = analogRead(34); //data from ADC signal (ecg electrodes)

bufferRawAndFiltered[bufferIndex] = data_RAW; // buffer with data raw
bufferRawAndFiltered[bufferIndex+7500] = Filter_For_Display(0, data_RAW); // buffer with the data filtered

bufferFilteredForDisplay[bufferIndex] = Filter_For_Display(0, data_RAW); // buffer with data with filter
bufferIndex++; // inc counter buffer
bufferIndex = bufferIndex%BufferSize; //limit in the index of the bufffer, so it does not grow ad infinitum
```

Abbildung 5-4 VS-Code: Sammlung und Speicherung von Daten

Das Array *bufferRawAndFiltered* (mit 15000 Positionen) hat 4 Flags, die sich jeweils im Viertel des Arrays befinden. Die flags werden wahr, wenn ihr Viertel gefüllt ist. Diese Flags werden von der Datei main.c verwendet, um die Daten an Matlab zu senden (s. Abb. 5-11). Der Code, der für diese Flags verantwortlich ist, lautet wie folgt:

```
if(bufferIndex == BufferSize/2) {
  flag1 = true; // first part of buffer raw is ready
  flag3 = true; // first part of buffer filtered is ready
  Serial.println("flag 1 Buffer Raw and filtered = true");
  Serial.println("flag 3 Buffer Raw and filtered = true");
}

if(bufferIndex == BufferSize-1) {
  flag2 = true; // second part of buffer raw is ready
  flag4 = true; // second part of buffer filtered is ready
  Serial.println("flag 2 Buffer raw and filtered = true");
  Serial.println("flag 4 Buffer raw and filtered = true");

}
```

Abbildung 5-5 VS-Code: Buffer and Flags

Bevor die daten in *bufferForDisplay* gespeichert werden, werden sie durch eine Filterfunktion namens *Filter_For_Display()* bearbeitet. Diese Funktion filtert die Frequenzen zwischen 49,5 Hz und 50,5 Hz, da durch 50 Hz im deutschen Stromnetz Störungen auftreten können. Die gefilterten Daten werden dann in *bufferForDisplay* gespeichert und danach auf dem OLED-Display übertragen, somit wird eine klare grafische Darstellung erzeugen.

Um diesen IIR (Infinite Impulse Response) Filter zu entwickeln, werden Koeffizienten benötig, die aus der Matlab-Anwendung „Filter Designer“ mit den erforderlichen Parametern berechnet wurden. (s. Abb. 5-5).

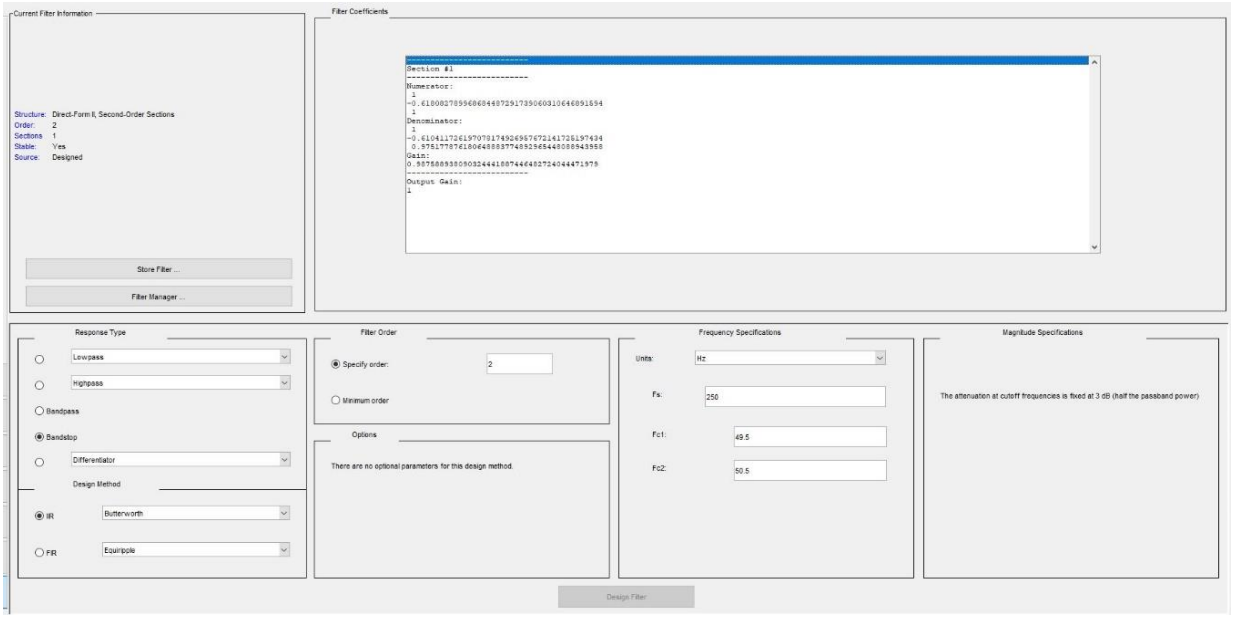


Abbildung 5-6 Matlab Filter Designer: Filter Koeffizient

Sobald die Koeffizienten ermittelt sind, kann ein Filter mit dem folgenden Abbildung 5-7 erstellt:

```
// global variables for the digital filter
double b0 = 1;
double b1 = -0.618082789968684487291739060310646891594;
double b2 = 1;
double a0 = 1;
double a1 = -0.610411726197078174926957672141725197434;
double a2 = 0.975177876180648883774892965448088943958;
double Mem0[1] = {0};
double Mem1[1] = {0};
double Mem2[1] = {0};
uint16_t data_RAW;
double data_flt;
double Filter_For_Display(int k, double x);

// digital filter
double Filter_For_Display(int k, double x) {
  double y;
  Mem0[k] = x - a1 * Mem1[k] - a2 * Mem2[k];
  y = b0 * Mem0[k] + b1 * Mem1[k] + b2 * Mem2[k];
  Mem2[k] = Mem1[k];
  Mem1[k] = Mem0[k];
  return y;
}
```

Abbildung 5-7 VS-Code: Digitaler IIR-Filter

Die Funktion *Filter_For_Display()* benötigt zwei Argumente: int k und double x. In das erste Argument (int k) setzen wir eine Null, um die Arrays an der ersten Position zu beginnen. In das zweite Argument (double x) geben wir den Wert der Quelle der Elektroden ein, der über die Funktion *analogRead()* in den Code gelangt. Die im Code verwendete Funktion sieht wie folgt aus:

```
data_RAW = analogRead(34);
Filter_For_Display(0, data_RAW);
```

Eine detaillierte Analyse des Filters findet sich in Kapitel 7.

5.4 EKG auf OLED (display.h)

Die Datei display.h enthält drei Funktionen: mean(), calc_value_for_display() und update_OLED(). Die Funktion mean() nimmt eine bestimmte Anzahl n von Werten (in ihrem Argument angegeben) und berechnet den Durchschnitt dieser Werte. Danach berechnet calc_value_for_display() die Position des nächsten Punkts auf der vertikalen Achse des OLED-Displays. Dazu ruft diese Funktion die Funktion mean() auf. Schließlich nimmt die Funktion update_OLED() den Wert, der von der Funktion calc_value_for_display() zurückgegeben wird, und druckt ihn auf dem Bildschirm aus. Dabei wird auch eine Linie zwischen diesem Punkt und dem vorherigen Punkt mit dem Befehl display.drawLine() gezeichnet.

```
inline void update_OLED() // current pixel position calculate
{

  static int oldy = 0;
  if (avantiPixelX == 0)
    {
      display.clear();//clear Display
      calc_value_for_display(oldy);
      display.setPixel(avantiPixelX, oldy); // draw current Point
    }else if(avantiPixelX > 0){
      int cury = 0;
      calc_value_for_display(cury);
      display.drawLine(avantiPixelX-1, oldy, avantiPixelX, cury);
      oldy = cury;
    }
    display.display();//display it
    avantiPixelX++;//next x
    avantiPixelX = avantiPixelX % SCREEN_WIDTH;

}
```

Abbildung 5-8 VS-Code: update_OLED()

5.5 EKG (main.c)

Anschließend enthält die datei main.c zwei Funktionen: setup() und loop(). In der Funktion void setup() wird das Display initialisiert. Der Mikrocontroller wird mit Hilfe der zuvor genannten WLAN_connection() mit dem WLAN verbunden und der Schalter wird konfiguriert. Für diesen letzten Teil werden die folgenden Codezeilen verwendet, wobei Einlesen die Schalterfunktion ist, die wir in der einlesen.h-Datei haben, und 8000 auf die 4 Millisekunden Abtastfrequenz verweist (s. Abb. 5-9).

```
  // Configure the interrupt
  AnalogTimer = timerBegin(0, 40, true);
  timerAttachInterrupt(AnalogTimer, &Einlesen, true);
  timerAlarmWrite(AnalogTimer, 8000, true); // 4 ms
  timerAlarmEnable(AnalogTimer);

  Serial.println("setup success: start matlab");
}
```

Abbildung 5-9 VS-Code: Konfiguration des Interrupts

In der Funktion void loop() wird ein MATLAB-Paket empfangen und in der Variablen packetSize gespeichert. Wenn es empfangen wurde, wird eine Do-While-Schleife gestartet, die das Senden von Daten vom Mikrocontroller zu MATLAB verwaltet. Im Folgenden zeigt man den Code in Matlab, der die Daten sendet, und den Code in Visual Studio, der sie empfängt.

```
void loop() {

  int packetSize = Udp.parsePacket();

  if (packetSize) {
    // receive incoming UDP packets
    Serial.printf("Received %d bytes from %s, port %d\n", packetSize, Udp.remoteIP().toString().c_str(), Udp.remotePort());
    int len = Udp.read(incomingPacket, 255);

    if (len > 0) {
      incomingPacket[len] = 0;
      matlab_start = true;
      Serial.println("Matlab starting");
    }
    Serial.printf("UDP packet contents: %s\n", incomingPacket);

    if (matlab_start==true) {
```

Abbildung 5-10 VS-Code: Erster Teil der Funktion loop()

Dieser Datenaustausch wird über die Flags gesteuert, die in einlesen.h deklariert und programmiert wurden.

Wenn flag1 „true“ ist, bedeutet dies, dass der erste Teil des Puffers voll ist und dieser Teil in 15 Paketen zu je 250 Bytes durch eine For-Schleife gesendet wird. Am Ende wird eine boolesche Variable true, wenn der Übertragungsprozess erfolgreich durchgeführt wurde.

```
do {

  if(flag1){ // when the first part of raw buffer is full

    Serial.println("start sending Part 1 of buffer raw ");
    for (int i = 0; i<=14; i++) { //loop 0 ... loop 14, so 15 loops
      // Send first part of bufferFiltered
      Udp.beginPacket(Udp.remoteIP(), Udp.remotePort()); //initializes the packet of data
      Udp.write((uint8_t*) &bufferRawAndFiltered[i*250], SEND_SIZE*2); //writes the message with data filtered
      sendSuccess1 = Udp.endPacket(); //send message

      if (sendSuccess1) {
        Serial.printf("     sending Packet %d \n", i);
      }
      if (!sendSuccess1) {
        Serial.printf("     ERROR sending Packet %d \n", i);
      }
      delay(100); // if there is no delay: [27031][E][WiFiUdp.cpp:185] endPacket(): could not send data: 12
    }

    Serial.println("end sending Part 1 of buffer raw");
    flag1 = false;
    Serial.println("flag 1 = false");
    sentPart1ofBufferAndFiltered = true;
    Serial.println("sentPart1ofBufferAndFiltered = true");
  }
```

Abbildung 5-11 Vs-Code: erster Teil der Do-While-Schleife

Wenn flag2 auf "true" gesetzt ist, deutet dies darauf hin, dass der zweite Teil des Puffers vollständig mit Daten gefüllt ist und dass der erste Teil des Puffers erfolgreich gesendet wurde., wird der zweite Teil auch in 15 Paketen zu je 250 Bytes gesendet. Am Ende wird eine boolesche Variable wahr, wenn der Übertragungsprozess erfolgreich durchgeführt wurde. Zu diesem Zeitpunkt wurden bereits 7500 Bytes gesendet, was 30 Sekunden der Datenerfassung der Elektroden entspricht.

Sobald flag3 auf "true" gesetzt wird, wird der dritte Teil des Puffers voll ist und wenn der zweite Teil erfolgreich gesendet wurde, wird dieser Teil in 15 Paketen zu je 250 Bytes durch eine for-Schleife gesendet. Am Ende wird eine boolesche Variable true, wenn der Übertragungsprozess erfolgreich durchgeführt wurde. Dieser Teil und der folgende Teil entsprechen den vom VSCode gefilterten Elektrodendaten.

Wird flag4 „true“, bedeutet dies, dass das vierte Teil des Puffers voll ist und wenn der dritte Teil erfolgreich gesendet wurde, wird auch das vierte Teil in 15 Paketen zu je 250 Bytes gesendet. Zu diesem Zeitpunkt wurden bereits die 15000 Bytes von der array *bufferRawAndFiltered* gesendet, was 30 Sekunden der Datenerfassung der Elektroden entspricht, jeweils 30 Sekunden ohne Filter und die gleichen 30 Sekunden mit Filter.

Nachdem alle vier Teile gesendet wurden, verlässt das Programm die Do-While-Schleife und das Display wird aktualisiert. Die folgenden Abbildung zeigt den Aufruf, der in Abschnitt 5-4 beschriebenen Funktion *update_OLED()* und das Echtzeit EKG-Signal auf dem Display.

```
    } while(!sentPart1ofBufferAndFiltered || !sentPart2ofBufferAndFiltered || !sentPart3ofBufferAndFiltered);
  }
}
//OLED Display update
currentMillis = millis();
delay(15); // in msec.
update_OLED();  // output on display
return;
}
```

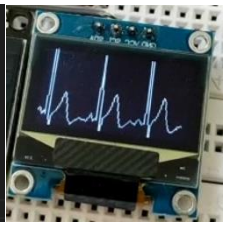


Abbildung 5-12 VS-Code: Letzter Teil der Do-While-Schleife und der loop() und Echtzeitablesung des EKG

6. EKG-Datenverarbeitung in Matlab

Um eine möglichst störungsfreie Darstellung der EKG-Daten zu erhalten, müssen die Daten von das 50Hz Netzbrummen gefiltert werden. Dazu wurden sowohl eine manuelle Filterung mittels einer Fouriertransformation mithilfe von Matlab eingesetzt. Die 7500 EKG-Daten wurden in Matlab in einem Array gespeichert und vom Zeitbereich in den Frequenzbereich transformiert. (s. Abb. 6-1)

```
% load data from file and save it into array
    A = load('ekg_array.mat');
    raw_ecg = A.ekg; % EKG-Daten als file speichern
    M = max(raw_ecg); % Für die Normierung
    raw_ecg = raw_ecg .* (1024/M); % von 0 bis 1024 Bits
    N = length(raw_ecg); %Anzahl der Elemente N = 7500

    fa = 250; %Abtastfrequenz Hz
    frq = linspace (0,fa,N); % Frequenzaxis
    t = linspace(0,N/fa,N+1);
    t = t(1:end-1);

    %Absolut des FFT des Signals Mit Normierung
    abs_FFT_ecg_filter = abs(fft(raw_ecg)/(0.5*N));
```

Abbildung 6-1 Matlab Code-Fragment: EKG-Signal Bearbeitung

Netzbrummen Filter

Um genauere Grafiken zu erhalten, wurde der erste Wert der Fourier-Transformation gefiltert, wo die Summe aller Frequenzen zu finden ist. Durch diesen Schritt wird deutlicher, wie sich die Einflüsse von Netzbrummen auf dem FFT-Plot in Matlab um 50 Hz und dessen Komplex (200 Hz) herumzeigen (s. Abb. 6-2).

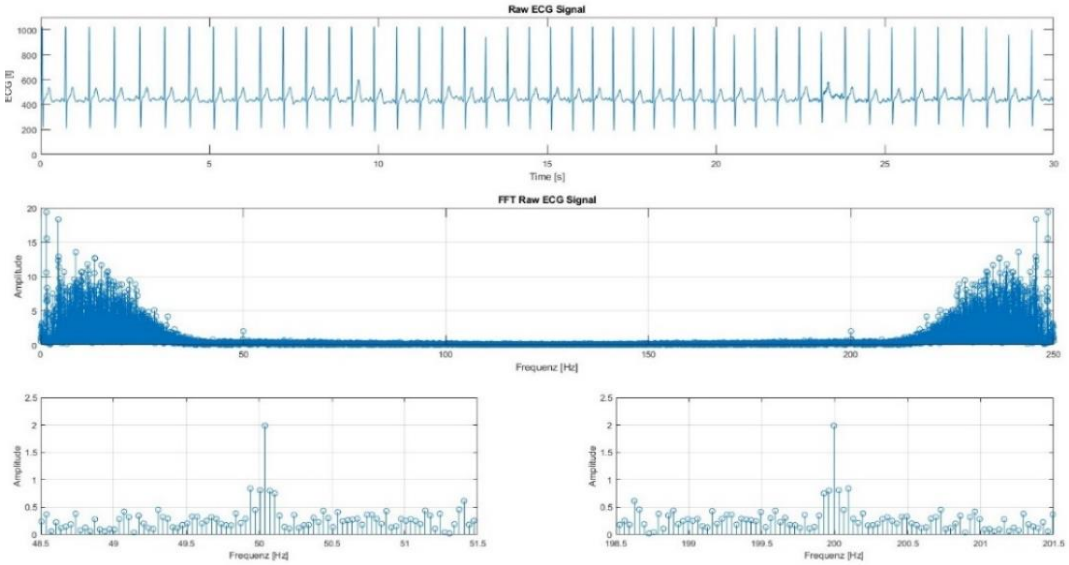


Abbildung 6-2 Matlab Plot: Raw EKG Signal (oben), FFT der Signal 1-250 Hz (Mitte), FFT Netzbrumme Peak (unten links) und FFT Netzbrumme PeakSpiegelung (unten rechts)

Die Elemente, die eine Frequenz zwischen 49,5 Hz und 50,5 Hz darstellten, wurden null gesetzt. Um diese zu bestimmen, war neben der zu eliminierenden Frequenz auch die Anzahl der Werte des Vektors und die daraus resultierende Abtastfrequenz notwendig (s. Abb. 6-3 und 6-4).

```
    FFT_ecg_filter = (fft(raw_ecg)); %FFT des Raw Signals

    f1=49.5;
    f2=50.5;

    f_index1  = round(f1*N/fa)+ 1;
    f_index1_2= N - f_index1 +1;    % Spiegelung im Frequenzspektrum
    f_index2  = round(f2*N/fa)+ 1;
    f_index2_2= N - f_index2 +1;    % Spiegelung im Frequenzspektrum

    FFT_ecg_filter (f_index1:f_index2) = 0;
    FFT_ecg_filter (f_index2_2:f_index1_2) = 0;

    abs_FFT_ecg_filter = abs(FFT_ecg_filter) ./ (0.5*N); % FFT Betrag mit
Normierung

    filter_ecg = ifft((FFT_ecg_filter));% Rücktransformation
```

Abbildung 6-3 Matlab Code-Fragment: EKG-Signal Filter

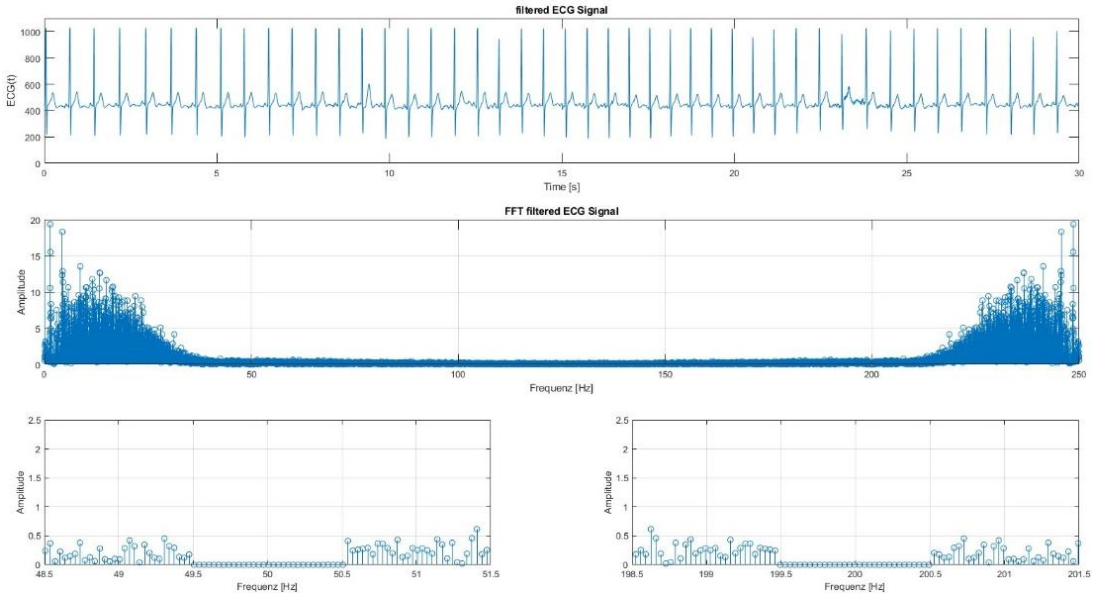


Abbildung 6-4Matlab Plot: Filtere EKG-Signal (oben), FFT der Signal 1-250 Hz (Mitte), FFT Netzbrumme Peak Filter (unten links) und FFT Netzbrumme Peak-Spiegelung Filter (unten rechts)

Es wurde festgestellt, dass die Form des Signals sich kaum verändert hatte und lediglich durch die inverse Fouriertransformation der gesamte Graph nach unten verschoben wurde. Das lag daran, dass sich kaum Frequenzen um 50 Hz befanden, wie der nächste Plot zeigt (s. Abb. 6-5).

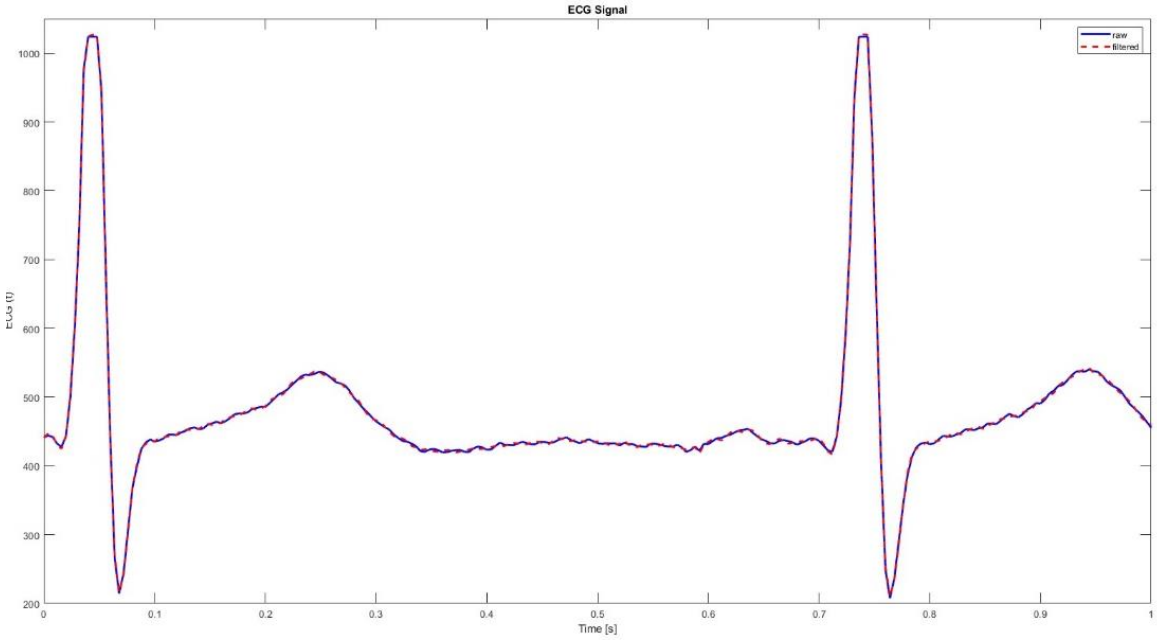


Abbildung 6-5 Matlab Plot: EKG-SIGNAL: Raw und Filter Vergleich

Detektion der R-Zacken in Matlab zur Analyse der Herzfrequenzvariabilität

Ein menschliches Herz schlägt nicht immer im perfekten Takt wie ein Metronom, sondern es gibt eine Variabilität in den Abständen zwischen den Herzschlägen. Diese Variabilität wird als Herzfrequenzvariabilität (HRV) bezeichnet und kann Aussagen über die Gesundheit des Herzens ermöglichen (luftpumper, 2021). Um die HRV zu analysieren, ist es wichtig, die QRS-Komplexe richtig zu untersuchen, insbesondere die R-Zacken.

Um die R-Zacken in Matlab zu zählen, müssen die Maxima einer Periode detektiert werden. Dabei wird ein Schwellenwert von 750 verwendet, um die P- oder T-Welle zu vermeiden. Ein Maximum wird erkannt, wenn der vorherige und der nachfolgende Wert kleiner sind als der aktuelle Wert. Nur wenn alle drei Bedingungen erfüllt sind, wird der Peak Counter **R_Ctr** erhöht. Dazu wird eine logische Vektor **TF** erzeugt, der eine 1 an den R Positionen speichert. Dieses ist hilfreich bei der R Zacken zu plotten.

Um die Herzfrequenz und die HRV zu ermitteln, werden die Zeitabstände zwischen zwei R-Zacken berechnet. Dafür wird im Matlab der Abstand in Elementen zwischen zwei detektierten R-Zacken berechnet. Die genauen Zeitpunkte, zu denen die R-Zacken auftreten, werden in das Array **R_pos** geschrieben. Der Zeitpunkt in Sekunden wird berechnet, indem der Index n durch die Frequenz (fa =250 Hz) geteilt wird. Ab dem zweiten Peak werden die Differenzen zwischen den einzelnen Zeitpunkten in ein weiteres Array **R_R** eingetragen (s. Abb. 6-6 und 6-7).

```
%% R-Zacken detektieren
    R_ctr = 0;   % Peak Counter
    R_pos = [];   % Peak Position
    R_R = [];  % Peak to Peak Distance
    TF = false(1,N); %in TF wird eine 1, wo die Positionen der R-Zacken sind

    for n=2:N-1
        if  (ECG(n)>ECG(n-1) && ECG(n)>ECG(n+1) && ECG(n)>750)
              R_ctr = R_ctr +1;
             R_pos = [R_pos, (n/fa)];
             % bei jedem Peak die Position aufschreiben
            TF(n) = true;
            if(R_ctr>=2)
                R_R = [R_R, (R_pos(R_ctr)-R_pos(R_ctr-1))];
            end
        end
    end

    %Plot ECG Signal With R

        plot(t,ECG,t(TF),ECG(TF),'r*')
        …
```

Abbildung 6-6 Matlab Code-Fragment: R-Zacken Bearbeitung (Detektion, Position und Abstand)

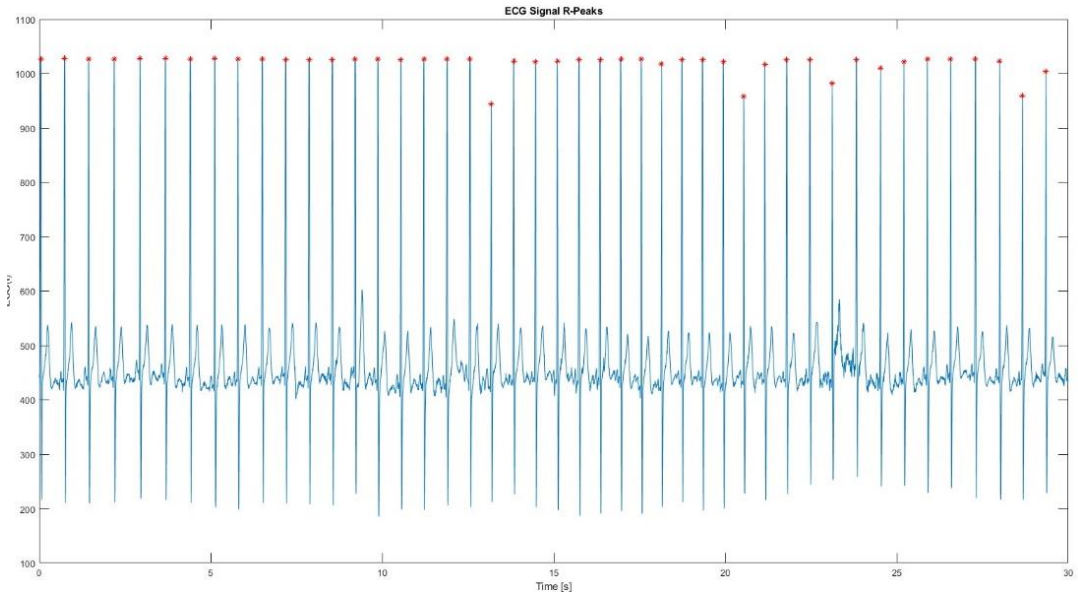


Abbildung 6-7 Matlab Plot: R-Zacken Detektion

Zunächst wird eine Schleife durchlaufen, um die Herzfrequenz (HR) für jedes R_R-Segment zu berechnen. Die Herzfrequenz wird berechnet, indem die Anzahl der Herzschläge pro Minute (BPM) durch Division von 60 durch die Dauer jedes R_R-Segments ermittelt wird. Der Wert für jede Herzfrequenz wird auf dem Bildschirm angezeigt.

Anschließend wird die Herzratenvariabilität (HRV) in Millisekunden berechnet, indem die Zeitdifferenzen zwischen den aufeinanderfolgenden R_R-Segmenten mit 1000 multipliziert werden. Die Herzratenvariabilität ist ein Maß für die Schwankungen der Herzfrequenz.

Zuletzt wird der RMSSD (Root Mean Square Successive Differences) in Millisekunden berechnet, der ein weiteres Maß für die Herzratenvariabilität ist. Der RMSSD wird berechnet, indem die quadrierten Differenzen zwischen aufeinanderfolgenden R_R-Segmenten gemittelt und die Wurzel des Ergebnisses gezogen wird. Der Wert für den RMSSD wird auf dem Bildschirm angezeigt.

```
%% Herzrate und Herzratenvariabilität berechnen
disp('Herzfrequenz');
for n = 1: length(R_R)
    HR(n) = 60/R_R(n);
    display(HR(n), 'BPM')
end

%Herzratenvariabilität in ms
HRV = diff(R_R)*1000;

% RMSSD in ms
disp('Herzratenvariabilität in ms');
%RMSSD = rms(HRV)
RMSSD = sqrt(mean(diff(R_R).^2))*1000;
display(RMSSD);
```

Abbildung 6-8 Matlab Code-Fragment: Herzrate und Herzvariabilität Berechnung

```
Herzfrequenz
BPM =

   86.2069

BPM =
   85.7143

BPM =
   80.6452
…
BPM =
   90.9091

BPM =
   87.7193

Herzratenvariabilität in ms

RMSSD =
   22.8809
```

Abbildung 6-9 Command Windows aus Matlab:Herzrate und Herzratenvariabilität Ausgabe

Insgesamt hat sich gezeigt, dass die Detektion der R-Zacken mit den beschriebenen Methoden zuverlässig funktioniert und wichtige Informationen für die Analyse der Herzfrequenzvariabilität liefert.

7. Filtern Vergleich

Der verbleibende Schritt ist die Analyse vom beide angewendete Filtern. Die Matlab (Methode 1) verwendet eine FFT-basierte Bandstop-Filterung, um Frequenzen um 50 Hz zu unterdrücken. Die VS CODE (Methode 2) verwendet einen IIR-Filter, um das Signal zu filtern. Die filtrierten Daten aus VS-Codes werden auch die über UDP gesendet und auf die gleiche Weise analysiert:

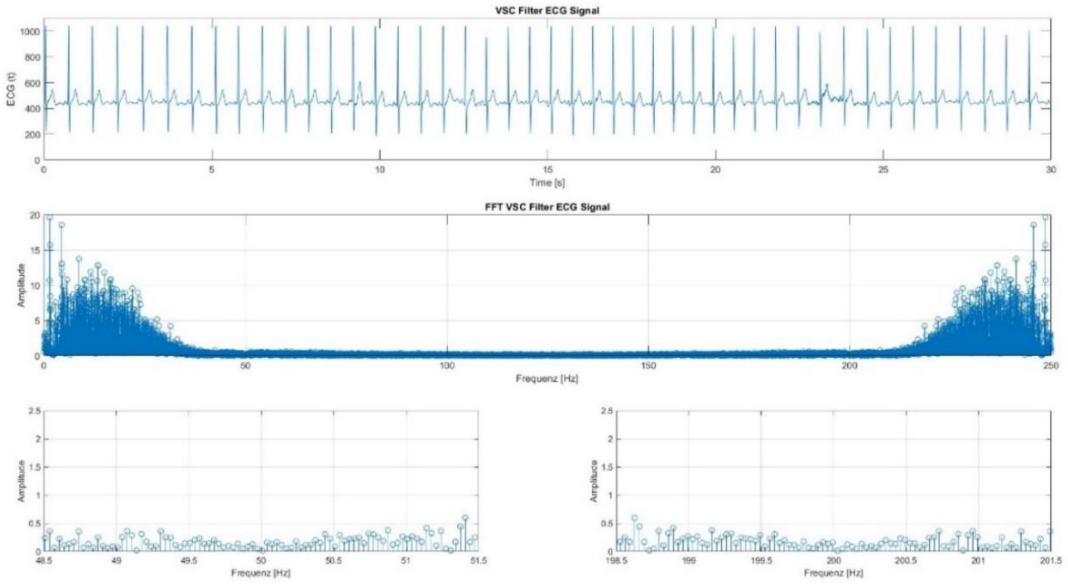


Abbildung 7-1 Matlab Plot: VSC filtrierte EKG-Signal (oben), FFT der Signal 1-250 Hz (Mitte), FFT Netzbrumme Peak Analyse (unten links) und FFT Netzbrumme Peak-Spiegelung Analyse (unten rechts)

Wie man sehen kann (s. Abb 7-1). Das empfangene Signal ähnelt im Großen und Ganzen dem mit der ungefilterten Funktion und dem von Matlab erzeugten Signal. Andererseits zeigt die Fourier-Transformation nicht die durch den Netzbrunnen verursachten Spitzen, die die ungefilterte Funktion hat, obwohl die Werte innerhalb des Filters nicht genau 0 sind. Somit kann behauptet werden, dass der Filter den Einfluss ziemlich stark reduziert, aber nicht vollständig.

Bei genauerer Analyse kann man erkennen, dass sich das von VS Code erzeugte Signal leicht von dem von Matlab gefilterten Signal unterscheidet, vor allem im P-, T- und U-Komplex gibt es kleine Sprünge im Signal. Ein weiterer Unterschied ist eine leichte Erhöhung der Spitzenwerte des QRS-Komplexes. (s. Abb. 7-2). Das bedeutet bei der Anwendung von unterschiedlichen Filtern auf das EKG-Signal, werden auch unterschiedliche Ergebnisse geliefert. Insbesondere weicht die Amplitude des gefilterten Signals in Methode 2 in einigen Bereichen vom Originalsignal ab und Methode 1. Dies kann auf die spezifischen Eigenschaften des IIR-Filters zurückzuführen sein, der das Signal verstärkt oder abschwächt, abhängig von der spezifischen Frequenz.

Um die Unterschiede zwischen den beiden Methoden genauer zu verstehen, es ist wichtig die Frequenzantworten der beiden Filter verglichen. Die Bandstop-Filterfrequenzantwort in Methode 1 zeigte eine totale Eliminierung der Frequenzen um 50 Hz, während der IIR-Filter in Methode 2 eine unterschiedliche Frequenzantwort aufwies, abhängig von den spezifischen Filterkoeffizienten.

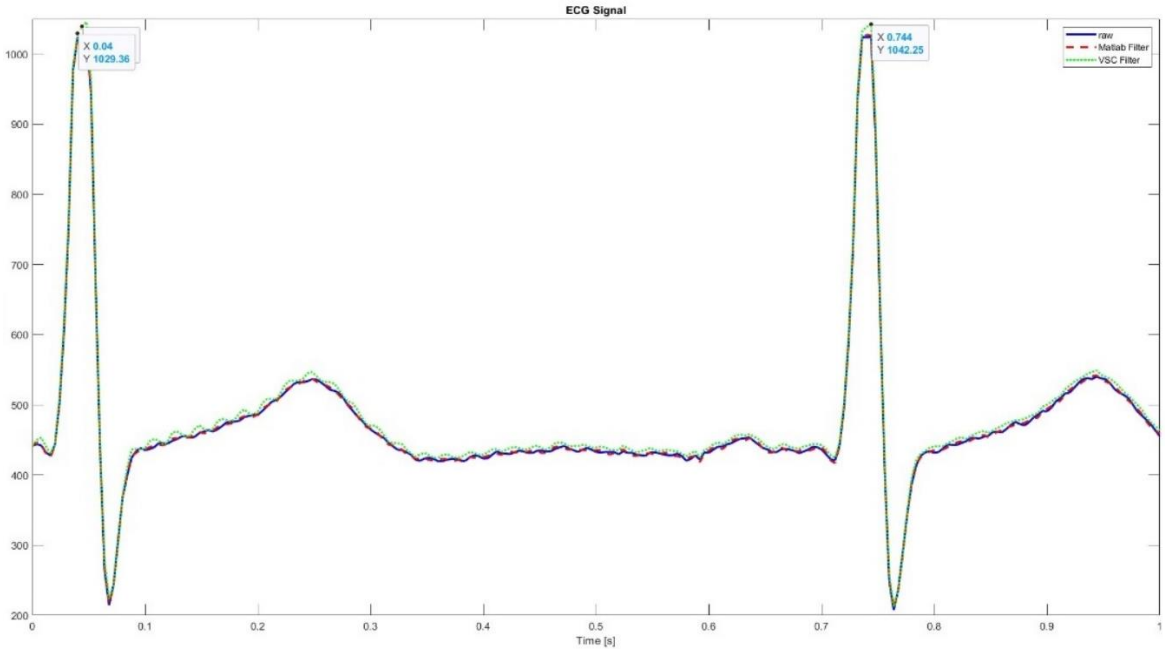


Abbildung 7-2 Matlab Plot: EKG Signale Vergleich

In Bezug auf die unterschiedlichen Maximalwerte der gefilterten Signale gibt es viele Faktoren, die dazu beitragen können, einschließlich der Art und Weise, wie das Filter auf das Signal angewendet wird und der Art des Filters, die in das Filter eingespeist werden.

8. Zusammenfassung und Ausblick

Das EKG-System wurde erfolgreich aufgebaut und getestet. Die Ergebnisse, die mit beiden Filtern erzielt wurden, erwiesen sich als zuverlässige Werte für EKG-Messungen. Die Echtzeit-Anzeige auf dem OLED-Display funktionierte einwandfrei und ermöglichte eine schnelle und genaue Überwachung des EKG-Signals.
Während des Projekts kam es zu Komplikationen/Verzögerungen, die Zeit, Tests und die Behebung von Fehlern erforderten, darunter die wichtigsten:

- (Vs Code) Entwicklung der Unterbrechung: mangelndes Verständnis für ihre Implementierung und Funktionsweise.
- (Vs Code) Die Übertragung von Daten durch die Funktion *loop()* an Matlab: korrekte Verwendung von Flags und Methode zum Senden von Informationspaketen.
- (Hardware) Falsche Verwendung von Ground: Aufgrund eines Druckfehlers im ESP wurden viel Zeit und Analyse sowohl der Software als auch der Hardware investiert, um zu verstehen, dass ein Druckfehler vorlag und die verwendete PIN nicht Ground war.

Die erfolgreiche Durchführung des EKG-Projekts war eine bereichernde Erfahrung und ermöglichte es, die im Studium erworbenen Kenntnisse in der Praxis anzuwenden. Es war eine Herausforderung, die Fähigkeiten und Tätigkeiten, die in der Arbeitswelt zu erwarten sind, auf die Probe zu stellen. Insgesamt war das Projekt ein erfolgreicher Beweis für die Anwendbarkeit der in der Vorlesung behandelten Themen in der Praxis und eine wertvolle Erfahrung für die zukünftige Arbeit im Bereich der Mechatronik und Informationstechnik.

9. Literaturverzeichnis

Analog Devices. (kein Datum). *www.analog.com.* Von https://www.analog.com/media/en/technical-documentation/data-sheets/ad8232.pdf abgerufen

AZ-Delivery. (kein Datum). Von www.az-delivery.de: https://www.az-delivery.de/es/products/1-3zoll-i2c-oled-display abgerufen

EWALD, W. (23. July 2023). *wolles-elektronikkiste.de*. Von https://wolles-elektronikkiste.de/en/programming-the-esp32-with-arduino-code abgerufen

luftpumper. (27. Mai 2021). */luftpumper.de.* Von https://luftpumper.de/herzfrequenzvariabilitaet/ abgerufen

Miron, R. (23. April 2020). *https://www.all-electronics.de/*. Von https://www.all-electronics.de/elektronik-entwicklung/wie-sich-ein-hochaufloesendes-ekg-entwickeln-laesst.html abgerufen

10. Abbildungsverzeichnis