

Datenbankbasierte Laufmaschine

Von

Manuel-Leonhard Rixen

Bearbeitungszeitraum:

04/16 - 12/16



Copyright

Manuel-Leonhard Rixen

Kurzfassung

Mithilfe eines speziellen Aufnahmesystems erfolgt das Erfassen definierter Bewegungsabläufe eines menschlichen Organismus. Aus diesen Werten werden parametrierbare Schrittfolgen extrahiert und in einer Datenbank abgespeichert. Ein bipedales Robotersystem greift über das Internet auf diese Daten zu und bewegt sich entsprechend der Schrittfolgen.

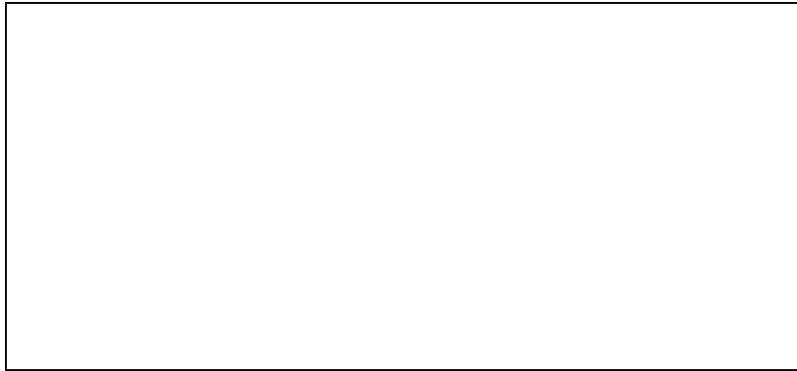


Abb. 0.1: Bipedale Laufmaschine

Inhalt

1 Einleitung	1
2 Messsystem	2
2.1 Einheit	2
2.1.1 Verwendete Komponenten	3
2.1.2 Testversion	5
2.1.3 Beta version	6
2.1.4 Release version	7
2.1.5 Programmaufbau	7
2.2 Hauptrechner	10
2.2.1 Verwendete Komponenten	10
2.2.2 Release Version	11
2.2.3 Programmaufbau	11
2.3 Anzug	11
2.3.1 Verwendete Komponenten	12
2.3.2 Testversion	12
2.3.3 Beta version	12
2.3.4 Release version	12
2.4 Erfassungsprogramm	13
2.4.1 Beschreibung HMI - Main	13
2.4.2 Beschreibung HMI - Visualisierung	15
2.4.3 Beschreibung HMI - Datenbank	16
2.4.4 Beschreibung Programmaufbau	17
3 Datenbank Layout	25
4 Robotersystem	26
4.1 Ansteuerung	26
4.1.1 Protokoll	28
4.1.2 Testprogramm	28
4.1.3 Kontrollprogramm	28
4.1.4 Clientprogramm	28
4.2 Mechanisches Modell	28
4.2.1 Prinzipieller Aufbau	28
5 Zusammenfassung	29

6 Ausblick	29
Anhangsverzeichnis	I
Abbildungsverzeichnis	IV
Tabellenverzeichnis	V
Erklärung	VI

1 Einleitung

Bereits vor dem 21. Jahrhundert besteht der Wunsch eine Maschine zu erschaffen, die sich durch ihre Agilität und Dynamik im menschlichen Lebensraum problemlos fortbewegen kann, um körperlich schwere und stumpfe Arbeiten dem Menschen abzunehmen. Als Vorbild dient der Mensch selbst, da sich dessen körperliche Struktur im Laufe der Evolution bewährt hat und, von der physischen Seite betrachtet, ausgereift ist.

Es existieren viele Arten von Humanoiden Robotern, durch die das Verständnis der Interaktion im menschlichen Lebensraum, sowie der menschlichen Fortbewegung intensiv erforscht wird.

Während viele Konstrukte wieder verworfen werden, bleiben andere bestehen, um diese kontinuierlich zu optimieren. Die Analyse und Bewertung der Systeme erfolgt u.A. anhand von Roboter-Events, bei denen verschiedene Tasks autonom ausgeführt werden müssen. Dabei zeigt sich, dass die Menschen ähnliche Fortbewegung zwar noch nicht ausgereift, jedoch in einem gewissen Bereich gut funktionstüchtig ist. Die technische Umsetzung erfolgt aus Regelalgorithmen, die in ihrem Umfang eingeschränkt sind.

Aus diesem Grunde wird in dieser Ausarbeitung ein datenbankbasierter Ansatz verfolgt. Das beinhaltet die Aufnahme der Bewegung beteiligter Gliedmaßen bei der menschlichen Fortbewegung. Daraus erfolgt das Extrahieren einer Datenbank, sowie das Implementieren eines Algorithmus, der mit diesen Daten die menschliche Fortbewegung nachbilden kann.

Der erste Teil besteht in dem Aufbau eines Messsystems, mit dem die Bewegungen aufgenommen und entsprechende Schrittfolgen extrahiert werden können (s. Kapitel 2). Im zweiten Teil erfolgt der Aufbau eines mechanischen Systems und die Entwicklung eines Movement-Control-Algorithmus (s. Kapitel 4):

2 Messsystem

Das Messsystem besteht aus mehreren Sensor-Einheiten, die an bestimmten Positionen eines menschlichen Körpers angebracht sind. Mit diesen Einheiten erfolgt das Erfassen von definierten Bewegungsabläufen, wie z.B. das Gehen geradeaus auf einer geraden Ebene, etc. Die Sensor-Einheiten sind über ein Bus-System (CAN-Bus) miteinander verbunden und kommunizieren mit einem Hauptrechner (Server), der die Sensordaten verarbeitet und über einen TCP-Socket versendet. Eine Client-Application verbindet sich mit dem Server und dient als HMI des Systems. Mit diesem Programm können die Messwerte grafisch dargestellt und in einer Datenbank abgespeichert werden.

Die Sensor-Einheiten und der Server sind in einen tragbaren Anzug eingearbeitet, während die Client-Application über einen herkömmlichen PC oder ein Smartphone bedient werden kann (Stoppen, Starten eines Messvorgangs, etc.).



Abb. 2.1: Messsystem

2.1 Einheit

Die Mcp-Executor-Einheit definiert ein Gerät, welches Beschleunigungsdaten über SPI an ein Master-Gerät schickt, sobald es eingeschaltet wird.

Für diese Funktionalität werden die in Kapitel 2.1.1 beschriebenen Komponenten verwendet, welche wie in Abbildung 2.2 dargestellt entsprechend verknüpft sind.

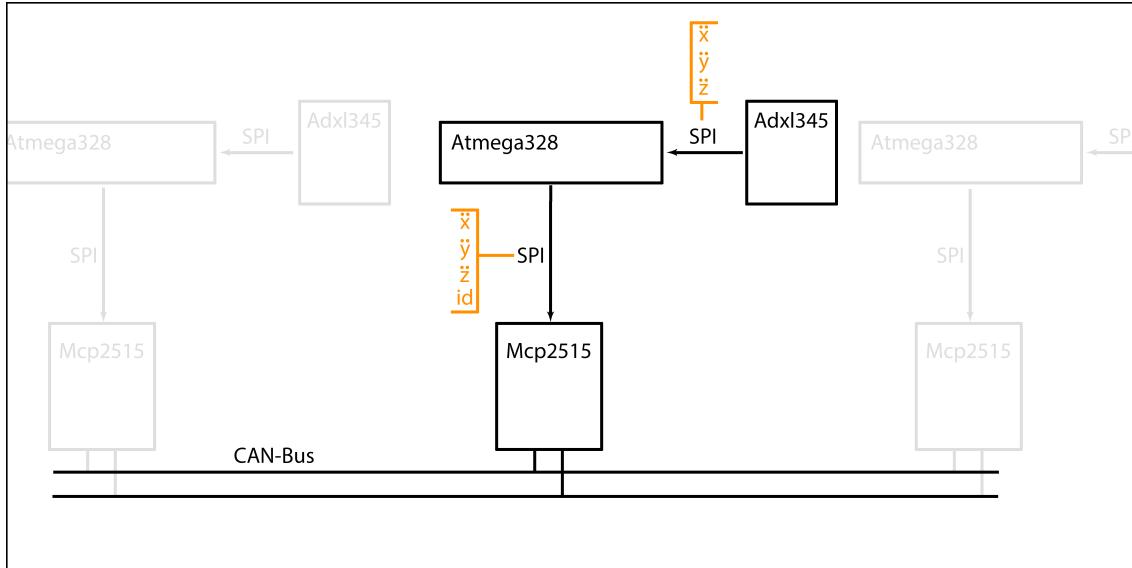


Abb. 2.2: Zusammenspiel der Komponenten Mcp-Executor-Einheit

Der Microcontroller Atmega328 lässt von dem Beschleunigungssensor Adxl über die Schnittstelle SPI Sensordaten aus und gibt diese (ebenfalls über SPI) an den Mcp2515 weiter. Dieser schiebt die Daten auf den CAN-Bus.

2.1.1 Verwendete Komponenten

Für eine Einheit werden verschiedene Hardware-Komponenten verwendet, deren Wahl vor allem hinsichtlich des Kostenaufwandes erfolgte. Auch die Bauteilgröße ist ein entscheidender Faktor, da die Einheiten möglichst nicht störend sein sollen.

- **Mcp2515**

Der Mcp2515 beschreibt an sich den Chip, welcher auf der, in Abbildung 2.3 dargestellten, Platine verbaut ist. Dieser Name wird jedoch für den gesamten Chip verwendet.

Die Funktion des Mcp2515 besteht darin Daten über die Schnittstelle SPI zu empfangen und über einen CAN-Bus zu versendet. Die Steuerung des Chips (z.B. Beschreiben eines Sende-Buffers) erfolgt dabei mit entsprechenden Kommandos über SPI. Das Konvertieren in L-, H-Pegel erfolgt komplett automatisch.

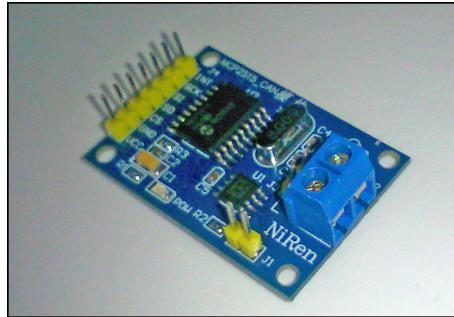


Abb. 2.3: Mcp2515

- **Adxl345**

Bei dem Adxl345 handelt es sich um einen 3-Achsen Beschleunigungssensor (s. Abbildung 2.4), dessen Messwerte über SPI in verschiedenen Genauigkeiten ausgelesen werden können. Der Chip besitzt eine äußerst geringe Leistungsaufnahme und ist kompatk in seiner Bauform.



Abb. 2.4: Adxl345

- **Atmega328**

Der Microcontroller Atmega328 ist u.A. in dem Arduino-Board verbaut. Es ist ein kostenfünstiger Allrounder, welcher hauptsächlich wegen seiner großen Verbreitung gewählt wurde. Der Chip (s. Abbildung 2.5) besitzt eine SPI-Schnittstelle und diverse I/O's, weshalb die Projektanforderungen vollständig erfüllt werden.

Es exisiteren zwei verschiedene Versionen des MCU's, von denen die Atmega328-P Variante für eine geringe Leistungsaufnahme konzipiert ist.

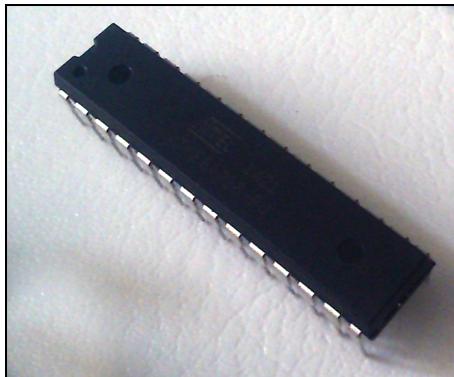


Abb. 2.5: Atmega328

- **Oscillator und Kondensator**

Der Atmega328 benötigt einen externen Taktgeber, welcher mit einem 16Mhz Quarz und entsprechenden Kondensatoren gewählt wird (s. Abbildung 2.6).



Abb. 2.6: Oscillator

2.1.2 Testversion

Um die Funktionen der Chips zu verstehen und beherrschen zu können erfolgt der Aufbau einer Einheit auf einem Entwicklungsboard (Whiteboard). Dieser Aufbau beinhaltet alles, was für eine Einheit vorgesehen ist, d.h. einen Atmega328 (mit externem Taktgeber), einen Adxl345 und einen Mcp2515. Mit entsprechender Verdrahtung (s. Schaltbild ??) und der Entwicklung eines Programms, welches auf der MCU aktiv ist, lassen sich die Sensorwerte über einen CAN-Bus auslesen. Die Sensor-ID muss programmatisch auf der MCU definiert werden.

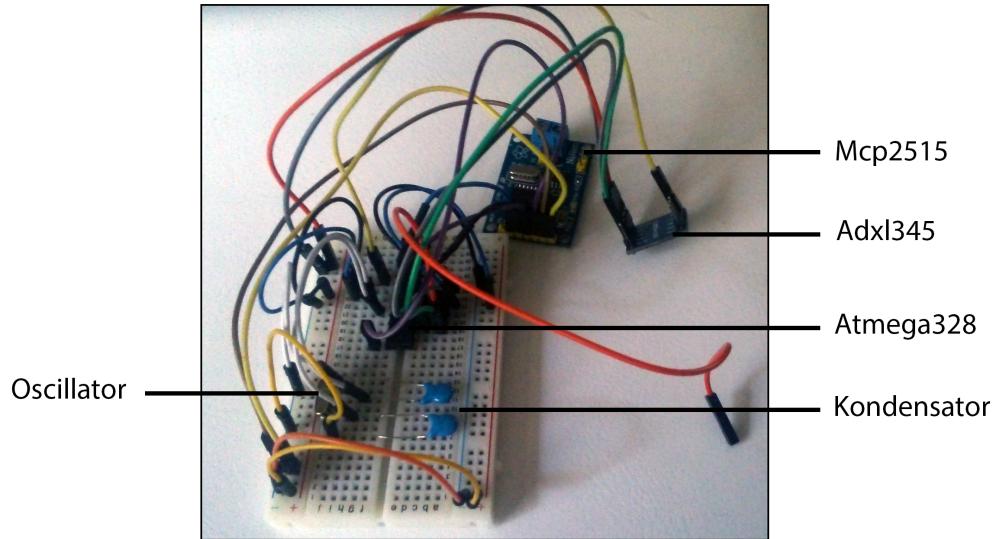


Abb. 2.7: Testversion einer Mcp-Executor-Einheit

Mit zwei solcher Testaufbauten wurde das Aufnehmen von Sensorwerten validiert.

2.1.3 Beta version

Für die Betaversion der Mcp-Executor-Einheit werden alle Komponenten in ein Gehäuse integriert (s. Abbildung 2.8), welches gerade so groß ist, dass es beim Tragen am Körper möglichst nicht stört und einfach befestigt werden kann. Für das Integrieren werden zusätzliche Platinen angefertigt.

Aufgrund des Feststellens verschiedener negativer Aspekte während der Entstehung der Einheiten existieren mehrere Beta-Versionen, von denen jedoch nur die erste aufgezeigt wird.

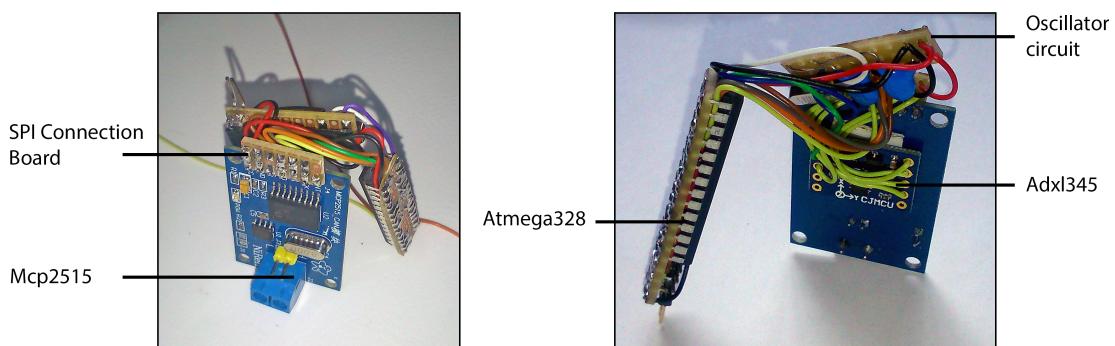


Abb. 2.8: Betaversion einer Mcp-Executor-Einheit

2 Messsystem

Die Befestigung der Einheit am Körper erfolgt anhand eines Positioniergurtes. Dieser besteht in der ersten Beta-Version aus zwei und in der zweiten Version aus einem Klettverschluss, die an dem Abdeckblech der Einheit befestigt sind. Durch ein entsprechendes Gegenstück des Klettverschlusses auf der Rückseite des Abdeckblechs ist der notwendig Halt gegeben.

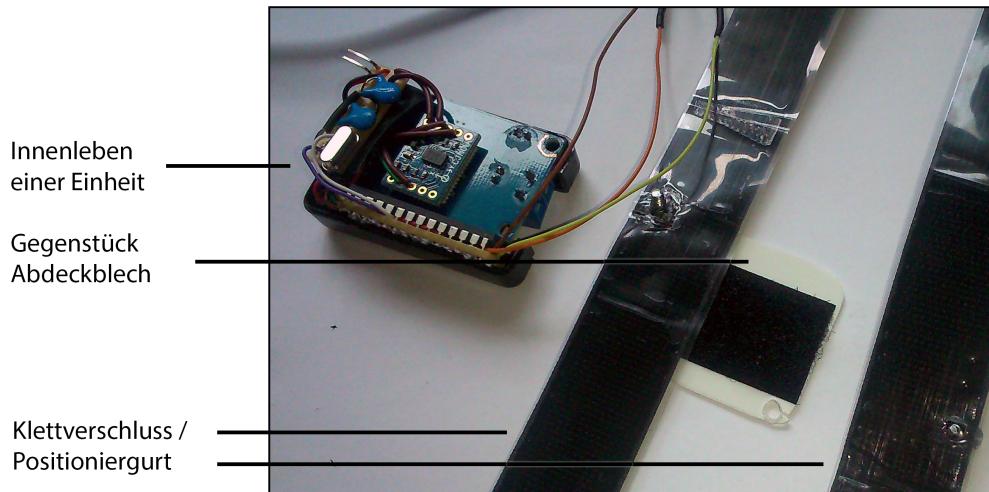


Abb. 2.9: Betaversion einer Mcp-Executor-Einheit - Klettverschluss

An dem Anzug sind ebenfalls entsprechende Gegenstücke des Klettverschlusses angebracht, sodass die Einheiten während der Bewegung nicht verrutschen.

2.1.4 Release version

Bei der Realease-Version der Mcp-Executor-Einheit wird die Beta-Version zunächst bewertet und alle negativen Aspekte aufgezeigt und mögliche Lösungen erarbeitet. Die zu verbessernden Punkte sind in Tabelle ?? aufgelistet:

2.1.5 Programmaufbau

1. Setup-Routine

Das Programm, welches auf der Micro Controller Unit (MCU) einer Einheit

aktiv ist besteht aus einer Setup- und einer Loop-Routine. Innerhalb des Setups erfolgt die Definition verschiedener Variablen und die Konfiguration der SPI-Schnittstelle, sowie der Komponenten Adxl345 und Mcp2515 (s. Listing 1). In den Zeilen 4-6 und 8-10 werden die Ausgänge zum Auswählen der Geräte, entsprechend der SPI-Schnittstelle, gesetzt. Dabei ist zu beachten, dass bei der Atmega-MCU der Ausgang an Pin 10 immer immer gesetzt werden muss, auch wenn ein anderer Pin als Device-Selector verwendet wird. Der Signalzustand der Pins muss HIGH entsprechen, da bei einem Low-Pegel das entsprechende Gerät ausgewählt wird.

Zeile 12-13 initialisiert die SPI-Schnittstelle mit 5 Mhz und dem SPI-Mode 3 (CPOL = 1, CPHA = 1). Diese einzige verfügbare Einstellung richtet sich nach dem Beschleunigungssensor Adxl345. Mit dem Aufruf der Routine SPI.begin() beginnt die Verwendung der Schnittstelle, sodass in den Zeilen 15-16 die Initialisierungsdaten an die Geräte gesendet werden können. Diese beinhalten für den Sensor Adxl345 u.A. das Datenformat und für den Mcp2515 das CAN-Bus Setup. Für das CAN-Bus Setup ist zu beachten, dass die Nachrichten jedes Teilnehmers einen spezifischen Identifier aufweisen muss.

Listing 2.1: Setup Routine

```

1 void setup()
2 {
3 ...
4 pinMode(CS_PIN_ADXL, OUTPUT);
5 pinMode(CS_PIN_MCP2515, OUTPUT);
6 pinMode(10, OUTPUT);
7
8 digitalWrite(CS_PIN_ADXL, HIGH);
9 digitalWrite(CS_PIN_MCP2515, HIGH);
10 digitalWrite(10, HIGH);
11
12 SPI.beginTransaction(SPISettings(5000000, MSBFIRST, SPI_MODE3));
13 SPI.begin();
14
15 initAdxl();
16 initMcp2515();
17 }
```

2. Loop-Routine

In der Loop-Routine (s. Listing 2) werden zwei Methoden aufgerufen, von denen die Prozedur *getAdxlData()* die Sensordaten liest und zusammen mit einem Zeitstempel in ein Array speichert. Die Routine *mcp2515_load_tx_buffer()* schreibt die Daten nacheinander in die Sende-Buffer 1-3.

Innerhalb der Routine *getAdxlData()* erfolgt in Zeile 10 das Setzen der entspre-

chenden Registerwerte, um die Sensordaten auslesen zu können. In den Zeilen 12-15 erfolgt das Auslesen, indem der Selektor-Pin auf Low-Level gelegt wird. Anschließend erfolgt das Senden von 0-Werten, um per SPI die Sensordaten auszulesen (für jede Achse zwei Byte). In Zeile 17 wird noch die ID für das Gerät angegeben.

Innerhalb der Routine mcp2515_load_tx_buffer0() erfolgt in den Zeilen 25-47 das Setzen der Registeradresse eines Sendebuffers, in Abhängigkeit welcher zuletzt genutzt wurde. In Zeile 51 werden in den ausgewählten Sendebuffer die Sensordaten gelegt und mit dem in Zeile 53 angegebenen Kommando auf den CAN-Bus geschickt.

Listing 2.2: Loop Routine

```

1 void loop()
2 {
3     getAdxlData();
4
5     for (size_t i = 0; i < MESSAGE_SIZE_ADXL; i++) mcp2515_load_tx_buffer(ReadBuf
6         [i], i, MESSAGE_SIZE_ADXL);
7 }
8 ...
9 bool getAdxlData() {
10
11     byte RegAddrBuf[] = { REGISTER_ACCEL_REG_X | REGISTER_ACCEL_SPI_RW_BIT |
12         REGISTER_ACCEL_SPI_MB_BIT };
13
14     setCsPin(CS_PIN_ADXL, LOW);
15     SPI.transfer(RegAddrBuf[0]);
16     for (int i = 0; i < 6; i++) ReadBuf[i] = SPI.transfer(0x00);
17     setCsPin(CS_PIN_ADXL, HIGH);
18
19     ReadBuf[6] = EXECUTOR_ID;
20
21 }
22 ...
23 void mcp2515_load_tx_buffer(byte messageData, int byteNumber, int messageSize)
24 {
25     byte registerAddress;
26
27     if (currentTxBuffer[0])
28     {
29         registerAddress = REGISTER_TXB0DX[byteNumber];
30         if (byteNumber == (messageSize-1))
31         {
32             currentTxBuffer[0] = false;
33             currentTxBuffer[1] = true;
34         }
35     }
36     else if (currentTxBuffer[1]) {
37         registerAddress = REGISTER_TXB1DX[byteNumber];
38         if (byteNumber == (messageSize - 1))
39         {

```

```
38     currentTxBuffer[1] = false;
39     currentTxBuffer[2] = true;
40   }
41 }
42 else if (currentTxBuffer[2]) {
43   registerAddress = REGISTER_TXB2DX[byteNumber];
44   if (byteNumber == (messageSize - 1))
45   {
46     currentTxBuffer[2] = false;
47     currentTxBuffer[0] = true;
48   }
49 }

51 mcp2515_execute_write_command(registerAddress, messageData, CS_PIN_MCP2515);

53 for (int i = 0; i < 3; i++) if(currentTxBuffer[i] == true)
54   mcp2515_execute_rts_command(i);
55 }
```

2.2 Hauptrechner

Der Hauptrechner definiert ein Gerät, welches als Master die Sensordaten aller Mcp-Executor-Einheiten anfordert und verarbeitet.

2.2.1 Verwendete Komponenten

Die Bestandteile des Hauptrechners sind ein Raspberry-Pi 2 mit Wlan-Adapter und dem Betriebssystem Windows 10 IoT, sowie einer Energiequelle (Akku von Anker) mit 20100 mAh.

- **Raspberry-Pi 2**

Der Raspberry Pi 2 (s. Abbildung ??) ist in seiner zweiten Version ein stabiles System, welches in Verbindung mit Windows IoT in der Hochsprache C# programmiert werden kann. Die Programme entsprechen den Anwendungen wie man sie von Smartphones kennt.

Die zweite Version beinhaltet (im Vergleich zur Version 3) keinen Wlan-Adapter, weshalb dieser zusätzlich erworben werden muss.

- **Wlan-Adapter**

Bei der Wahl eines Wlan-Adapters muss die Kompatibilität zu Windows-IoT berücksichtigt werden, da nicht für jedes Gerät entsprechende Treiber zur

Verfügung stehen. Aus diesem Grunde liegt die Wahl bei dem preisgünstigen Wlan-Adapter XXXX (s. Abbildung 2.4).

- **Akku von Anker**

Für die Energieversorgung (s. Abbildung ??) wird ein Akku benötigt, welcher eine hohe Kapazität aufweist und genügend Strom liefert, damit das System ordnungsgemäß funktioniert. Bspw. benötigt der Raspberry Pi 2 2A, um einen fehlerfreien Betrieb zu gewährleisten (mit zusätzlich angeschlossener Hardware, wie z.B. der Wlan-Adapter).

Der Akku PowerCore 20100 von Anker ist für diese Anforderungen wie geschaffen, da dieser 20100 mAh aufweist und 2 USB-Ports mit jeweils 2A zur Verfügung stellt.

2.2.2 Release Version

Pseudotext Pseudotext Pseudotext Pseudotext Pseudotext Pseudotext Pseudotext
Pseudotext Pseudotext Pseudotext Pseudotext Pseudotext Pseudotext Pseudotext

2.2.3 Programmaufbau

Pseudotext Pseudotext Pseudotext Pseudotext Pseudotext Pseudotext Pseudotext
Pseudotext Pseudotext Pseudotext Pseudotext Pseudotext Pseudotext Pseudotext

2.3 Anzug

Pseudotext Pseudotext Pseudotext Pseudotext Pseudotext Pseudotext Pseudotext
Pseudotext Pseudotext Pseudotext Pseudotext Pseudotext Pseudotext Pseudotext
Pseudotext Pseudotext Pseudotext Pseudotext Pseudotext Pseudotext Pseudotext

Pseudotext Pseudotext Pseudotext Pseudotext Pseudotext Pseudotext Pseudotext
Pseudotext Pseudotext Pseudotext Pseudotext Pseudotext Pseudotext Pseudotext
Pseudotext Pseudotext Pseudotext Pseudotext Pseudotext

2.3.1 Verwendete Komponenten

2.3.2 Testversion

2.3.3 Beta version

2.3.4 Release version

Pseudotext Pseudotext Pseudotext Pseudotext Pseudotext Pseudotext Pseudotext
Pseudotext Pseudotext Pseudotext Pseudotext Pseudotext Pseudotext Pseudotext
Pseudotext. Pseudotext Pseudotext Pseudotext Pseudotext Pseudotext Pseudotext

Pseudotext Pseudotext Pseudotext Pseudotext Pseudotext Pseudotext Pseudotext
Pseudotext Pseudotext Pseudotext Pseudotext Pseudotext Pseudotext Pseudotext
Pseudotext Pseudotext Pseudotext Pseudotext Pseudotext

2.4 Erfassungsprogramm

Das Lesen, bzw. Speichern der Sensordaten erfolgt mit einem Programm, welches sich als Client mit dem Server verbindet und über WLAN die Daten aller Sensoren empfängt. Die Sensorwerte werden nicht automatisch gespeichert, jedoch empfangen und können grafisch dargestellt werden (x-, y-, z-Werte für jeden Sensor in einer Übersicht). Ist das Speichern der Daten gewünscht, so muss eine Anzahl an Samples angegeben und der Button *Record to db* betätigt werden. Das Speichern der Daten erfolgt dann in eine lokale SQL-Datenbank.

2.4.1 Beschreibung HMI - Main

Die HMI gliedert sich in mehrere Bereiche, in denen verschiedene Funktionalitäten bereitgestellt werden. Diese finden im folgenden Abschnitt Erläuterung.

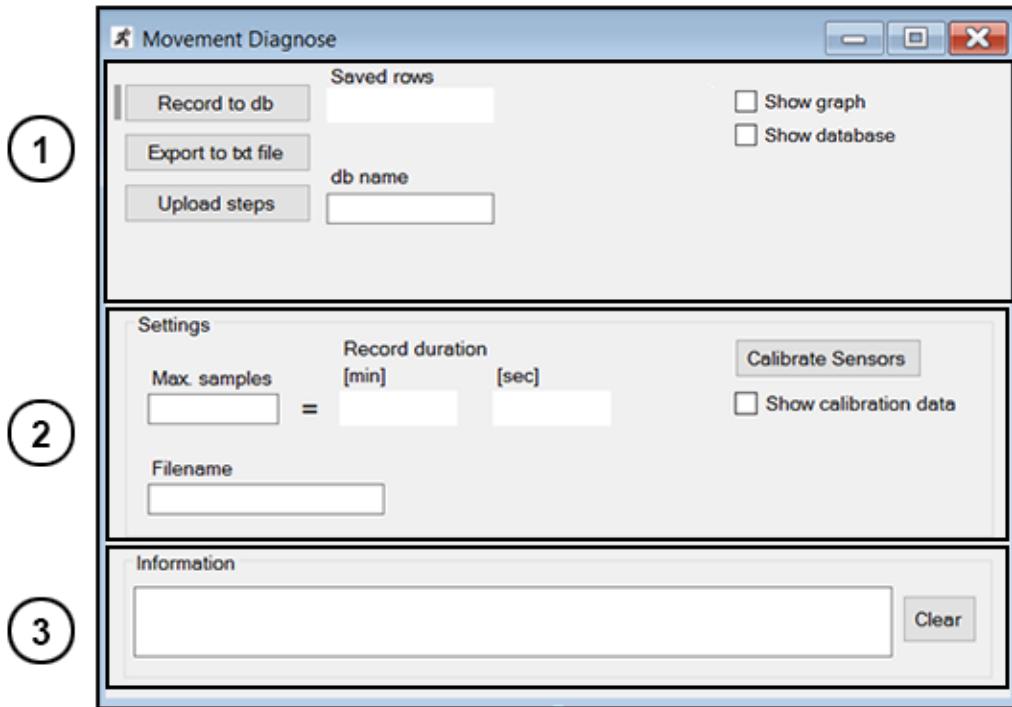


Abb. 2.10: Client Graph HMI

1. Aufnahme / Import, Export / Visualisierung

Der grüne Balken dient als Indikator, ob Messwerte empfangen werden (grün: Messung kann gestartet werden, rot: Es werden keine Daten empfangen).

Durch das Betätigen des Buttons *Record to db* wird eine Messung gestartet bis diese abgebrochen oder die max. Anzahl an Samples erreicht wird.

Das Label *Saved rows* zeigt die aktuelle Anzahl aufgenommener Messwerte an. Durch das Betätigen des Buttons *Upload steps* erfolgt das Importieren einer CSV-Datei (ohne Überschrift) in das Programm, welches diese dann in eine externe SQL-Datenbank hochlädt. Im Textfeld *db name* muss die Bezeichnung der Tabelle eingegeben werden, in welche die Messwerte hochgeladen werden sollen.

Mit dem Betätigen des Buttons *Export...* erfolgt das Exportieren der erfassten Sensordaten in eine CSV-Datei. Jeder Sensor generiert eine eigene Datei mit dem unter *Context* angegebenen Namen.

Diese Funktion soll in der nächsten Version automatisiert im Hintergrund stattfinden, sodass es, zum Hochladen in die externe Datenbank, nur ein Button existiert.

Durch das Setzen des Häkchens bei *Show graph* werden die Sensorwerte visualisiert. Durch das Setzen des Häkchens bei *Show database* werden die Einträge der Datenbank dargestellt.

2. Settings

In das Textfeld *Max. samples* wird die aufzunehmende Anzahl an Messwerten eingetragen. Danach stoppt die Messung automatisch. Standardmäßig sind 3500 Samples eingetragen, da bei einer zu großen Anzahl eine *Out of memory Exception* geworfen wird.

Durch das Betätigen des Buttons *Calibrate Sensors* erfolgt das Kalibrieren aller Sensoren (s. Beschreibung in 2.4). Durch das Setzen des Häkchens *Show...* erfolgt das Darstellen der kalibrierten Sensorwerte.

Für das Exportieren der Messwerte in eine Text-Datei kann ein Dateiname angegeben werden. Bleibt dieses Feld leer, so lautet der Dateiname: noname_x (x steht für die Zahl des Sensors).

3. Information

In dem Textfeld erfolgt das Darstellen wichtiger Systemereignisse (z.B. Fehler), die durch das Betätigen des Buttons *Clear* gelöscht werden können.

2.4.2 Beschreibung HMI - Visualisierung

Durch das Betätigen des Buttons *Show graph* innerhalb der Hauptansicht des Programms, erfolgt das Darstellen der Visualisierung.

Das Visualisieren der Sensorwerte erfolgt durch vier Diagramme pro Sensor (s. Abbildung 2.11), in denen die Beschleunigungswerte in jeder Achse einzeln und einmal alle zusammen dargestellt werden. Die Auswahl des Sensors erfolgt durch einen Up-Down-Selector (s. Abbildung - Pos. 2) und das Erstellen eines Abbilds durch das Betätigen des Buttons *Snapshot*. Dieses wird auf dem Desktop abgelegt.

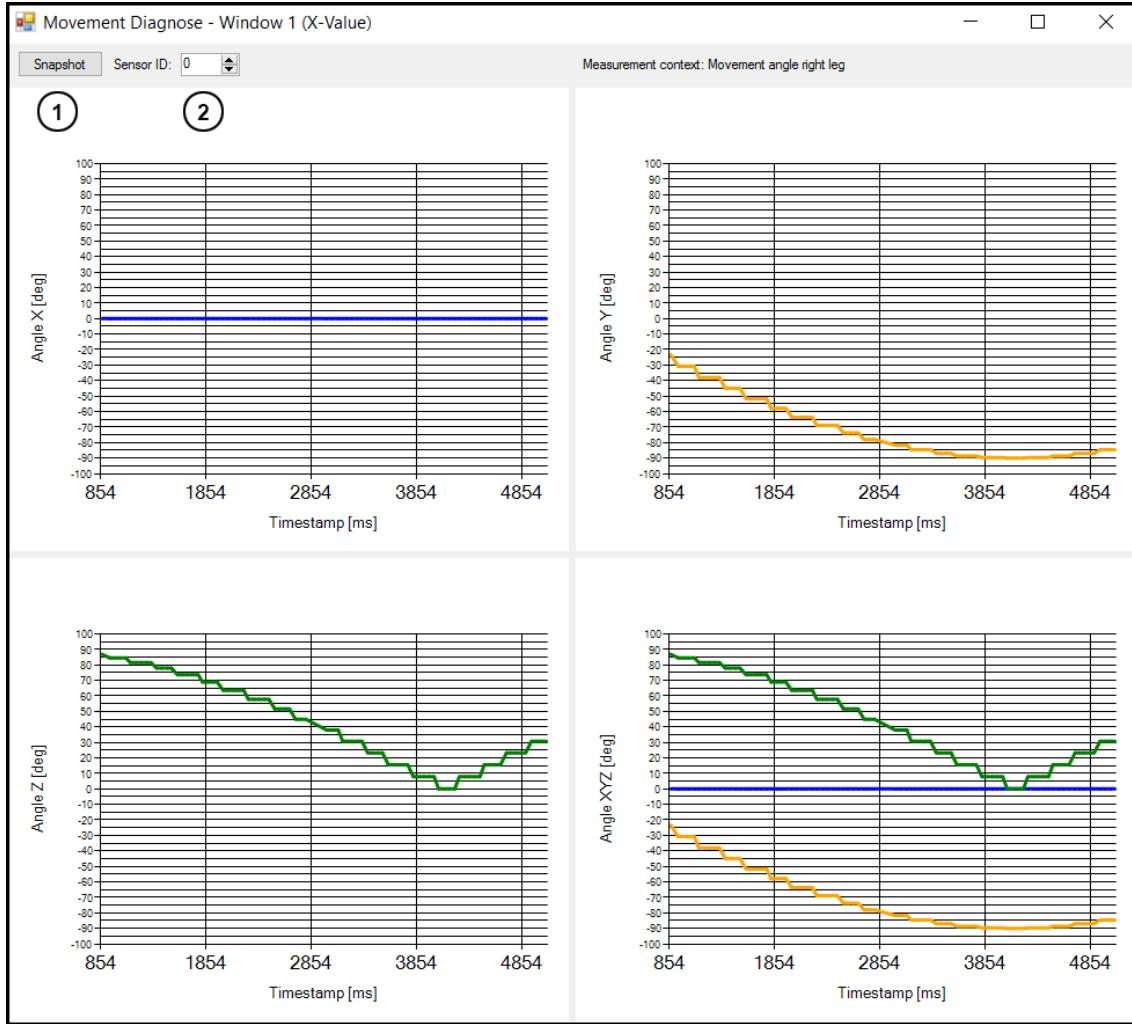
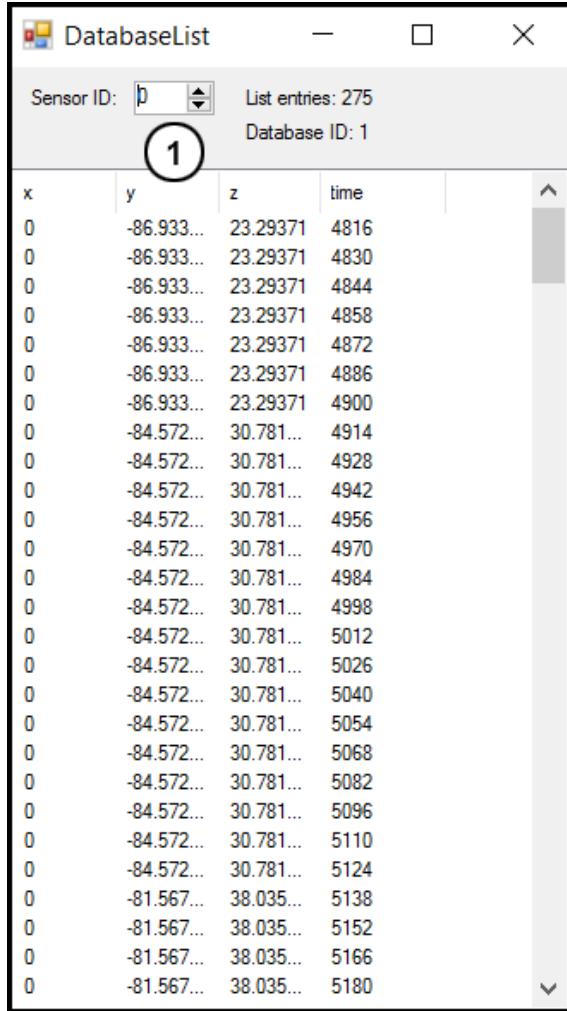


Abb. 2.11: Diagramm der Sensorwerte

2.4.3 Beschreibung HMI - Datenbank

Während des Aufzeichnens von Sensorwerten erfolgt das Abspeichern in eine Datenbank. Diese beinhaltet die Daten für jeden Sensor einzeln mit den Beschleunigungswerten in x, y und z, sowie einen Zeitstempel. Die Datenbank kann durch das Betätigen des Buttons *Show Database*, in der Hauptansicht des Programms, angezeigt werden (s. Abbildung 2.12).

Bei einer erneuteten Messwertaufnahme wird der Datenbankinhalt gelöscht.



x	y	z	time
0	-86.933...	23.29371	4816
0	-86.933...	23.29371	4830
0	-86.933...	23.29371	4844
0	-86.933...	23.29371	4858
0	-86.933...	23.29371	4872
0	-86.933...	23.29371	4886
0	-86.933...	23.29371	4900
0	-84.572...	30.781...	4914
0	-84.572...	30.781...	4928
0	-84.572...	30.781...	4942
0	-84.572...	30.781...	4956
0	-84.572...	30.781...	4970
0	-84.572...	30.781...	4984
0	-84.572...	30.781...	4998
0	-84.572...	30.781...	5012
0	-84.572...	30.781...	5026
0	-84.572...	30.781...	5040
0	-84.572...	30.781...	5054
0	-84.572...	30.781...	5068
0	-84.572...	30.781...	5082
0	-84.572...	30.781...	5096
0	-84.572...	30.781...	5110
0	-84.572...	30.781...	5124
0	-81.567...	38.035...	5138
0	-81.567...	38.035...	5152
0	-81.567...	38.035...	5166
0	-81.567...	38.035...	5180

Abb. 2.12: Datenbank mit Sensorwerten

2.4.4 Beschreibung Programmaufbau

In diesem Kapitel wird zu jeder Funktion (z.B. das Starten einer Messung) die dahinter liegenden Programm-Elemente aufgezeigt und erläutert. Triviale Funktionalitäten, wie z.B. das Löschen eines Textfeldes, werden dabei außer Acht gelassen.

Die Elemente werden dabei mit der in der HMI angegebenen Bezeichnung beschrieben.

1. Record to db

Das Starten einer Messwertaufnahme erfolgt durch das Betätigen des Buttons *Record to db*. In der dazugehörigen Event-Routine (s. Listing 1 Zeile 1-19) erfolgt zunächst eine Abfrage der Button-Beschriftung (Zeile 3), da nur ein Element zum Stoppen / Starten einer Messwertaufnahme genutzt wird. In

dieser Abfrage wird ein Dialog eingeblendet, der das Löschen des bisherigen Datenbankinhaltes veranlasst (Zeile 5-9).

Das Löschen der Datenbank erfolgt innerhalb eines Background-Tasks (s. Zeile 23-32). Hierbei wird in Zeile 26 und 27 die Datenbank 1 und 2 mit den entsprechenden Connectionstrings gelöscht. In Zeile 28 und 29 erfolgt das Erstellen der neuen Datensätze für Datenbank 1 und 2. Zeile 30 gibt den Datensatz 1 an die Routine *x_RunWorkerCompleted()* weiter. In dieser Routine wird die boolsche Variable *recordIsActive* gesetzt, sodass in der Event-Routine *tcpMsgRecEvent()*, die beim Eintreffen eines Messwerts aufgerufen wird, Werte in die Datenbank geschrieben werden können. Sobald die Variable wieder zurückgesetzt ist, erfolgt kein weiteres Schreiben in die Datenbank.

Listing 2.3: Messwertaufnahme starten

```

1  private void recordToDb_Click(object sender, EventArgs e)
2  {
3      if (button_recordToDb.Text.Equals("Record to db"))
4      {
5          if (MessageBox.Show("Delete database content?", "Database re-initialization
6              ", MessageBoxButtons.OKCancel, MessageBoxIcon.Question) ==
7                  DialogResult.OK)
8          {
9              labelSavedRows.Text = "";
10             backgroundWorker_DeleteDb.RunWorkerAsync();
11         }
12     else ; // Do nothing
13   }
14 ...
15     recordIsActive = false;
16     savedRowCounter = 0;
17 ...
18   }
19 ...
20 ...
21 private void backgroundWorker_DeleteDb_DoWork(object sender, DoWorkEventArgs e)
22 {
23 ...
24     databaseConnection.deleteDatabaseContent(ConnectionString_DB1);
25     databaseConnection.deleteDatabaseContent(ConnectionString_DB2);
26     dataSet_Db1 = databaseConnection.createDatasetsForDb(ConnectionString_DB1);
27     dataSet_Db2 = databaseConnection.createDatasetsForDb(ConnectionString_DB2);
28     e.Result = databaseConnection.getTableSizeForDb(dataSet_Db1);
29 ...
30   }
31 ...
32 private void backgroundWorker_DeleteDb_RunWorkerCompleted(object sender,
33             RunWorkerCompletedEventArgs e)
34 {
35 ...
36     recordIsActive = true;

```

```

36     maxTableRows_Db1 = (int[]) e.Result;
37 }
```

2. Export db to txt file

Das Exportieren der gespeicherten Messwerte (Formatierung: CSV-Datei, Trennzeichen: Semikolon) erfolgt durch das Betätigen des Buttons *Export db to txt*. In der dazugehörigen Event-Routine wird dazu ein Background-Task gestartet (Zeile 1-16). In dieser Routine wird zunächst die maximale Zeilenanzahl, sowie die Überschrift für das Textdokument gesetzt.

In Zeile 7-15 erfolgt das Schreiben der Daten in eine Textdatei. Diesbezüglich wird für jeden Sensor eine separate Datei mit spezifischen Dateinamen erstellt. Das Setzen des Dateinamens erfolgt im Textfeld *Context* innerhalb der Main-Gui. Wird kein Name angegeben, so tragen die Dateien die Bezeichnung *noname_x.txt*.

Da während des Schreibvorgangs alle Buttons deaktiviert werden, erfolgt in der Routine *...RunWorkerCompleted()* das Re-Aktivieren der entsprechenden Elemente.

Listing 2.4: Datenbank exportieren

```

1  private void backgroundWorker_saveDbToTxt_DoWork(object sender, DoWorkEventArgs
2      e)
3  {
4      ...
5      int[] maxTableRows = databaseConnection.getTableSizeForDb(dataSet_Db1);
6      string header = "x[deg];y[deg];z[deg];timestamp[ms]";
7
8      for (int i = 0; i < maxTableRows.Length; i++)
9      {
10         string fileName;
11         if (textBox_context.Text != "") fileName = textBox_context.Text + "_" + i +
12             ".txt";
13         else fileName = "noname_" + i + ".txt";
14
15         using (StreamWriter writer = new StreamWriter(FILE_SAVE_PATH + fileName,
16             false)) writer.WriteLine(header);
17         saveDbToFile(i, maxTableRows, fileName);
18     }
19 }
20 ...
21 private void backgroundWorker_saveDbToTxt_RunWorkerCompleted(object sender,
22     RunWorkerCompletedEventArgs e)
23 {
24     helperFunctions.changeElementEnable(button_exportToTxt, true);
25     helperFunctions.changeElementEnable(button_recordgetDb, true);
26     helperFunctions.changeElementEnable(button_importgetDb, true);
27 }
```

3. Upload steps

Zunächst muss in das Textfeld *db name* die Bezeichnung der Tabelle eingegeben werden, in die etwas importiert werden soll (Tabellenname in der externen SQL-Datenbank - s. Beschreibung Kapitel 3). Durch das Betätigen des Buttons erscheint dann ein Dialog, in dem der Bediener eine CSV-Datei (ohne Überschrift) auswählen muss. Diese Textdatei muss folgendes Format des Dateinamens aufweisen: tableX.txt (X steht für die Zahl des Sensors, z.B. 0, 1, 2, etc.).

Anschließend erfolgt das Hochladen in eine externe SQL-Datenbank. In einer späteren Version wird es nur noch einen Button geben, mit dem die Schrittfolge automatisch extrahiert und in die externe Datenbank hochgeladen wird.

Listing 2.5: Schrittfolge auslagern

```

1  private void button_uploadSteps_Click(object sender, EventArgs e)
2  {
3      ...
4      try
5      {
6          fileId = Int32.Parse(fileName[0]);
7          db_name = textBox_dbTableName.Text;
8      }
9      ...
10     finally
11     {
12         if (((fileId >= MIN_TABLE_AMOUNT) && (fileId <= MAX_TABLE_AMOUNT)) && (
13             db_name.Length != 0)) backgroundWorker_loadTxtToDb.RunWorkerAsync();
14         else helperFunctions.changeElementText(textBox_Info, "Wrong file id or db
15             name", true);
16     }
17 }
18 private void backgroundWorker_loadTxtToDb_DoWork(object sender, DoWorkEventArgs
19     e)
20 {
21     ...
22     xampp_removeContent(db_name, fileId);
23     using (StreamReader reader = new StreamReader(directoryTxtImport))
24     {
25         while (!reader.EndOfStream)
26         {
27             returnString = reader.ReadLine();
28             String[] messageData = returnString.Split(';');
29             Decimal[] messageDataAsDecimal = new Decimal[messageData.Length];
30
31             for (int j = 0; j < 4; j++) messageDataAsDecimal[j] = Decimal.Parse(
32                 messageData[j], CultureInfo.InvariantCulture.NumberFormat);
33             if (readCounter > 0) xampp_addContent(db_name, fileId,
34                 messageDataAsDecimal);
35             ...
36         }
37     }
38 }
```

4. Calibrate Sensors

Das Starten der Kalibrierung erfolgt durch das Betätigen des Buttons *Calibrate Sensors*. In der dazugehörigen Event-Routine (s. Listing 4 Zeile 1-9) werden zunächst die Kalibrierwerte (Zeile 5), sowie der Merker, für das Durchführen einer Kalibrierung (Zeile 6), zurückgesetzt.

In Zeile 8 wird der Button deaktiviert und erst wieder aktiviert wenn die Kalibrierung abgeschlossen ist.

Das Ausführen der Kalibrierung erfolgt in der Event-Routine *tcpMsgRecEvent()* (Zeile 11-52), die beim Eintreffen eines Messwerts ausgeführt wird. Ist der Button *Calibrate Sensors* deaktiviert und der Merker für die durchgeführte Kalibrierung *sensorCalibrationSet[]* gesetzt erfolgt das Kalibrieren in den Zeilen 18-36. Dabei wird zu Beginn in den Zeilen 18-20 der Quadrant geprüft, in dem sich das Koordinatensystem (KS) des Sensors befindet. Demnach erfolgt ein Abbruch, wenn ein Sensor sich nicht innerhalb der zulässigen Quadranten (s. Abbildung 2.13) befindet.

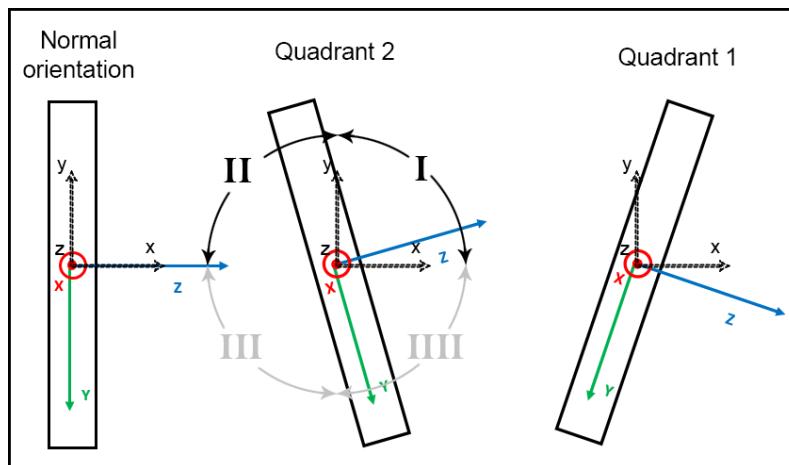


Abb. 2.13: Zulässige Quadranten

Listing 2.6: Sensoren kalibrieren

```

1  private void button_calibrateSensors(object sender, EventArgs e)
2  {
3      for (int i = 0; i < SENSOR_AMOUNT; i++)
4      {
5          gs_x[i] = 0; gs_y[i] = 0; gs_z[i] = 0;
6          sensorCalibrationSet[i] = false;
7      }
8      buttonCalibrateSensors.Enabled = false;
9  }
10 ...
11 private void tcpMsgRecEvent(string[] receivedMessage)
12 {
13     ...
14     if (sensor_joint_ID <= SENSOR_AMOUNT)

```

```

15    {
16        if ((!sensorCalibrationSet[sensor_joint_ID]) & (!buttonCalibrateSensors.
17            Enabled))
18        {
19            double alpha = -9999;
20            if ((sensorValues[1] >= 0) & (sensorValues[2] < 0)) alpha = -(Math.PI /
21                2) + Math.Acos((sensorValues[1] * GRAVITATION_EARTH) /
22                GRAVITATION_EARTH);
23            else if ((sensorValues[1] >= 0) & (sensorValues[2] > 0)) alpha = (Math.PI /
24                2) - Math.Acos((sensorValues[1] * GRAVITATION_EARTH) /
25                GRAVITATION_EARTH);
26
27            if (alpha != -9999)
28            {
29                Rx_x[0] = 1;
30                Rx_x[1] = 0;
31                Rx_x[2] = 0;
32
33                Rx_y[0] = 0;
34                Rx_y[1] = Math.Cos(alpha);
35                Rx_y[2] = -Math.Sin(alpha);
36
37                Rx_z[0] = 0;
38                Rx_z[1] = Math.Sin(alpha);
39                Rx_z[2] = Math.Cos(alpha);
40
41                sensorCalibrationSet[sensor_joint_ID] = true;
42
43                for (int i = 0; i < SENSOR_AMOUNT; i++)
44                {
45                    if (!sensorCalibrationSet[i]) break;
46                    else if (i == SENSOR_AMOUNT - 1) helperFunctions.changeElementEnable(
47                        buttonCalibrateSensors, true);
48                }
49            }
50        }
51        ...
52    }

```

5. Show graph

Das Visualisieren der Messwerte erfolgt innerhalb eines Hintergrund-Tasks. Diesbezüglich wird der Graph aktualisiert, sobald ein neuer Messwert empfangen wurde. Beim Empfangen eines Messwerts erfolgt der Aufruf der Event-Routine *tcpMsgRecEvent()* (s. Listing 5 Zeile 1-4). Innerhalb dieser Routine wird geprüft, ob die Checkbox zum Anzeigen der Visualisierung gesetzt und ob die Windows Form bereits deklariert ist. Ist dies der Fall, so erfolgt die

Übergabe aller Messwerte, sowie der Sensor-ID an die Routine *setNewChartData()* (s. Zeile 3).

In dieser Routine wird zunächst geprüft, ob die ausgewählte ID (mit dem Up- Down-Selector) mit der übertragenen Sensor-ID übereinstimmt. Ist dies der Fall so erfolgt das Setzen eines Events *chartData()* (s. Zeile 8). In der Event-Routine *OnChartEvent()* wird zunächst geprüft, ob die darzustellenden Daten bereits in dem Daten-Array vorhanden sind (Prüfen gegen Vorinitialisierung -9999). Ist dies der Fall so wird die Routine *setDataToGraph()* aufgerufen (Übergabewert: Sensordaten). In dieser Routine ist das Visualisieren der Messwerte implementiert (s. Zeile 16-24). Dargestellt ist der Code für das Setzen der Sensorwerte auf die X-Achse. Zeile 18-19 zeigt das Setzen der Achsbeschriftung und Zeile 21-22 zeigt das Setzen eines Messpunkts.

Listing 2.7: Visualisierung anzeigen

```

1 private void tcpMsgRecEvent(string[] receivedMessage)
2 {
3     if ((checkBox_showGraphs.Checked) && (formCharts != null)) formCharts.
4         setNewChartData(messageDataAsDecimal, sensor_joint_ID);
5 }
6
6 public void setNewChartData(Decimal[] message, int currentSensorID)
7 {
8     if (sensorIdToShow == currentSensorID) this.chartData = message;
9 }
10
11 private void OnChartEvent(object sender, EventArgs e)
12 {
13     if ((chartData[0] != -9999) && (chartData[1] != -9999) && (chartData[2] !=
14         -9999) && (chartData[3] != -9999)) setDataToGraph(chartData);
15     dataCounter++;
16 }
17
18 private void setDataToGraph(Decimal[] chartData)
19 {
20     chartX.ChartAreas[0].AxisX.Title = "Timestamp [ms]";
21     chartX.ChartAreas[0].AxisY.Title = "Angle " + chartXaxisLabel[0] + " [deg]";
22     ...
23     chartSeries[0].Points.AddXY(chartData[3], chartData[0]);
24     chartX.Invalidate();
25     ...
26 }
```

6. Show database

Das Darstellen der Datenbank, indem sich die aufgezeichneten Messwerte befinden, erfolgt durch das Setzen der entsprechenden Checkbox. Das Prüfen, ob die Checkbox gesetzt ist erfolgt innerhalb der verbundenen Event-Routine (s. Listing 6 Zeile 4). Entsprechend der Datenbank-ID wird die Datenbank aus-

gewählt und eine Liste mit dessen Inhalt erzeugt (s. Zeile 6) und dargestellt (s. Zeile 7).

Listing 2.8: Datenbank anzeigen

```
1 private void checkBox_showDatabase_CheckedChanged(object sender, EventArgs e)
2 {
3     ...
4     if (((CheckBox)sender).Checked)
5     {
6         DataBaseList = new DataBaseList(this, dataSet_Db1, databaseConnection,
7             databaseId);
8         DataBaseList.Show();
9     }
10    else try DataBaseList.Close();
11    ...
12 }
```

3 Datenbank Layout

Es existieren insgesamt zwei Systeme (Mess- und Robotersystem), von denen jedes über eine lokale Datenbank verfügt. Darüber hinaus existiert eine externe, über das Internet erreichbare Datenbank, in der die gesamten Bewegungsdaten für den Roboter hinterlegt sind. Diese sind als Integer-Werte abgelegt und mit dem Faktor 100 multipliziert, sodass diese Daten vom Prinzip unmodifiziert sind und ein breites Anwendungsgebiet aufweisen. Das Konvertieren in die nutzbaren Stellgrößen für das entsprechende System (in dem Fall das Robotersystem) erfolgt lokal.

Abbildung 3.1 zeigt diesen Sachverhalt auf.

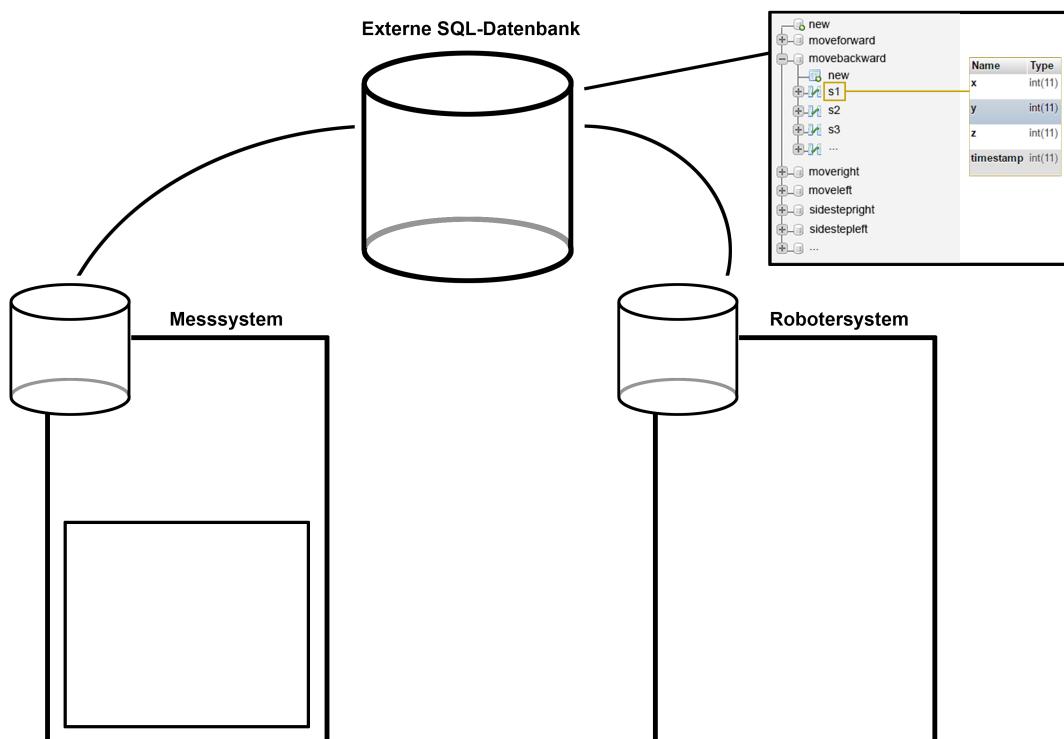


Abb. 3.1: Layout der Datenbanken

4 Robotersystem

Das Robotersystem beschreibt das Endprodukt dieses Projekts - eine mechanische Einheit mit bipedaler Fortbewegung. Diesbezüglich ist zu beachten, dass keinerlei Autonomie und Intelligenz in dem System steckt. Der Schwerpunkt liegt einzig auf der datenbankbasierten Fortbewegung.

Um eine bestmögliche Bewegungsfreiheit zu gewährleisten, muss das mechanische System perfekt mit dem des Menschen übereinstimmen. Nur dann sind dem Aufzeichnen, sowie dem Konvertieren in Bewegungsfolgen keine Grenzen gesetzt.

In diesem Projekt wird das Robotersystem mit manueller Eingabe bewegt (gesteuert). Die entsprechenden Softwarekomponenten sind diesbezüglich im Kapitel ?? beschrieben.

Das mechanische Modell findet im Kapitel 4.2 Erläuterung.

4.1 Ansteuerung

Der Aufbau des Gesamtsystem und dessen Funktionsweise ist in Abbildung XXX prinzipiell dargestellt.

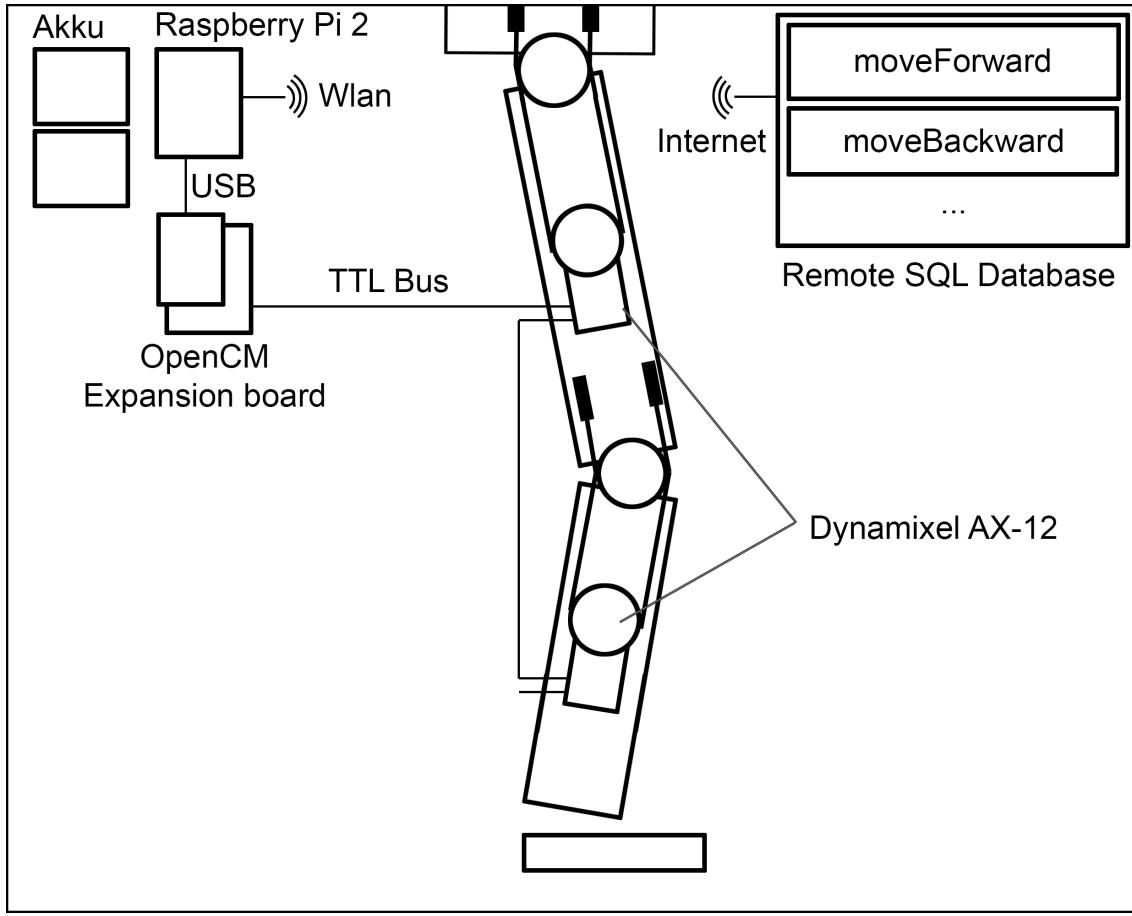


Abb. 4.1: Ansteuerung / Aufbau (Prinzip)

Dabei ist zunächst das mechanische Modell vernachlässigbar, da dieses im weiteren Verlauf Erläuterung findet. Zunächst wird die softwaretechnische Seite betrachtet und dessen elektronischen Komponenten, die zur Ansteuerung des Gesamtsystems genutzt werden, erläutert.

Diesbezüglich befindet sich im oberen linken Teil der GRafik die Konstellation der beteiligten Baugruppen und deren Verknüpfung zueinander. Der Raspberry Pi 2 dient als Steuerzentrale, auf dem das Betriebssystem (BS) Windows 10 IOT installiert ist. Darauf ist eine Software aktiv, mit der die gesamte Reglung des Systems abgearbeitet wird. Entsprechend erfolgt das Senden von Befehlen (zur Ansteuerung der Motoren) vom Raspberry Pi aus über eine USB-Schnittstelle hin zum OpenCM Board. Dieses Board liest die Befehle und gibt diese an die Motoren (Dynamixel AX-12) weiter. Das Expansion Board ist notwendig, da das OpenCM-Board keine TTL-Schnittstelle zur Verfügung stellt.

Der eigentlich Clou des Systems besteht darin, dass beim Neustart (sofern notwendig) die Bewegungssätze von einer externen Datenbank heruntergeladen werden. Diesbezüglich verfügt der Raspberry Pi über eine Wlan, bzw. UMTS-Schnittstelle.

Die SQL-Datenbank ist über das Internet erreichbar, sodass stets aktuelle Bewegungssätze zur Verfügung stehen. Die Bewegungssätze werden mit dem, in Kapitel 2 beschriebenen Messsystem erstellt / erweitert.

4.1.1 Protokoll

4.1.2 Testprogramm

4.1.3 Kontrollprogramm

4.1.4 Clientprogramm

4.2 Mechanisches Modell

4.2.1 Prinzipieller Aufbau

5 Zusammenfassung

Da eine künstliche Intelligenz zu unvorhersehbaren, gefährlichen Situationen führen kann, muss ein, dem Menschen ähnlich dynamisches System, in seiner Freiheit beschränkt werden. Der Begriff Freiheit definiert in diesem Kontext das eigenständige Lernen ohne menschliche Kontrolle.

Ein datenbankbasiertes System besitzt hingegen nur die Fähigkeiten, welche in der Datenbank hinterlegt sind. Dies bringt den Vorteil der Berechenbarkeit mit sich.

6 Ausblick

Stark unausgereift hingegen ist das zusammenhängende Verarbeiten visueller Eindrücke, auditiver Wahrnehmungen, sowie vestibulären Werten und sensomotorischen Fähigkeiten.

Anhangsverzeichnis

A.1 Listings	II
A.1.1 Listing 1	II
A.1.2 Listing 2	III

A.1 Listings

A.1.1 Listing 1

Listing A.1: NAME

```
1 public void Function {  
2  
3 }
```

Listing A.2: NAME

```
1 public void Function {  
2  
3 }
```

A.1.2 Listing 2

Listing A.3: NAME

```
1 public void Function {  
2  
3 }
```

Listing A.4: NAME

```
1 public void Function {  
2  
3 }
```

Abbildungsverzeichnis

0.1	Bipedale Laufmaschine	2
2.1	Messsystem	2
2.2	Zusammenspiel der Komponenten Mcp-Executor-Einheit	3
2.3	Mcp2515	4
2.4	Adxl345	4
2.5	Atmega328	5
2.6	Oscillator	5
2.7	Testversion einer Mcp-Executor-Einheit	6
2.8	Betaversion einer Mcp-Executor-Einheit	6
2.9	Betaversion einer Mcp-Executor-Einheit - Klettverschluss	7
2.10	Client Graph HMI	14
2.11	Diagramm Sensorwerte	16
2.12	Datenbank mit Sensorwerten	17
2.13	Zulässige Quadranten	21
3.1	Layout der Datenbanken	25
4.1	Ansteuerung / Aufbau (Prinzip)	27

Tabellenverzeichnis

Erklärung

Hiermit erkläre ich an Eides Statt, dass ich die vorliegende Arbeit selbstständig und ohne Benutzung anderer als der angegebenen Hilfsmittel angefertigt habe. Die aus fremden Quellen direkt oder indirekt übernommenen Gedanken sind als solche kenntlich gemacht.

ORT, DATUM



Manuel-Leonhard Rixen