Jörg Bernau (15411098)

Theo Kiesel

TECHNISCHE HOCHSCHULE
OSTWESTFALEN-LIPPE
UNIVERSITY OF
APPLIED SCIENCES
AND ARTS

Informatik im Maschinenbau (VIM2)

Serielle Datenkommunikation zwischen Matlab App und einer MCU

Abstract:

Diese Arbeit erläutert den Zusammenhang und Aufbau eines Versuchs zur rotatorischen Beschleunigung und dessen Widerstandsmomentes. Die Verwendung eines Schrittmotors in der mechanischen Versuchsapparatur und die Anbindung via Arduino mit custom Firmware ermöglicht die Kontrolle aller Parameter via Modbus. Der Versuch kann von einer Widget-basierten Matlab-App per GUI kontrolliert werden. Versuchsdaten werden dabei vollautomatisch erzeugt, aufgezeichnet und visuell für den Nutzer aufbereitet. Die asynchrone Modbus-Kommunikation zwischen App und Controller ließ sich im Rahmen dieses Projektes nicht fehlerfrei realisieren. Auftretenden Fehler wurden aber eingehend analysiert und Lösungsvorschläge entwickelt.

Inhalt

Einleitung

Inspiriert von den Versuchen aus der Veranstaltung *Messtechnik Grundlagen* desselben Dozenten, und der beruflichen Erfahrung in der Anwendung von LabView entstand die Idee eines vollautomatischen Versuchsaufbaus. In der Veranstaltung von *Informatik im Maschinenbau 2* wurde die visuelle Datenaufbereitung von Daten behandelt. Daher entstand die Idee Den Appdesigner von Matlab zu versenden, um einen Versuch damit aufzubauen.

Aus persönlicher Motivation des Autors die physikalischen Grundlagen der rotatorischen Beschleunigung zu durchdringen, entstand dieser Versuchsaufbau.

Aus dem Autor bekannten Automationsumfelds wurde das Protokoll des Modbus zur Steuerung einer selbst erstellen Platine übernommen. Über eine zweite serielle Schnittstelle gelangen die Messdaten zum PC und damit in die Matlab Applikation.

Widerstands- und Drehmomentmoment

Für die Translation gilt zwischen der Kraft $\vec{F}$, der Masse m und der Beschleunigung $\vec{a}$ der grundlegende Zusammenhang $\vec{F} = m \cdot \vec{a}$, das newtonsche Grundgesetz. Für die Rotation starrer Körper gilt analog das Grundgesetz der Dynamik der Rotation: $\overrightarrow{M} = J \cdot \vec{\alpha}$, . Wird mit $J = m \cdot r^2$ substituieret ergibt sich $\overrightarrow{M} = m \cdot r^2 \cdot \vec{\alpha}$: [1]

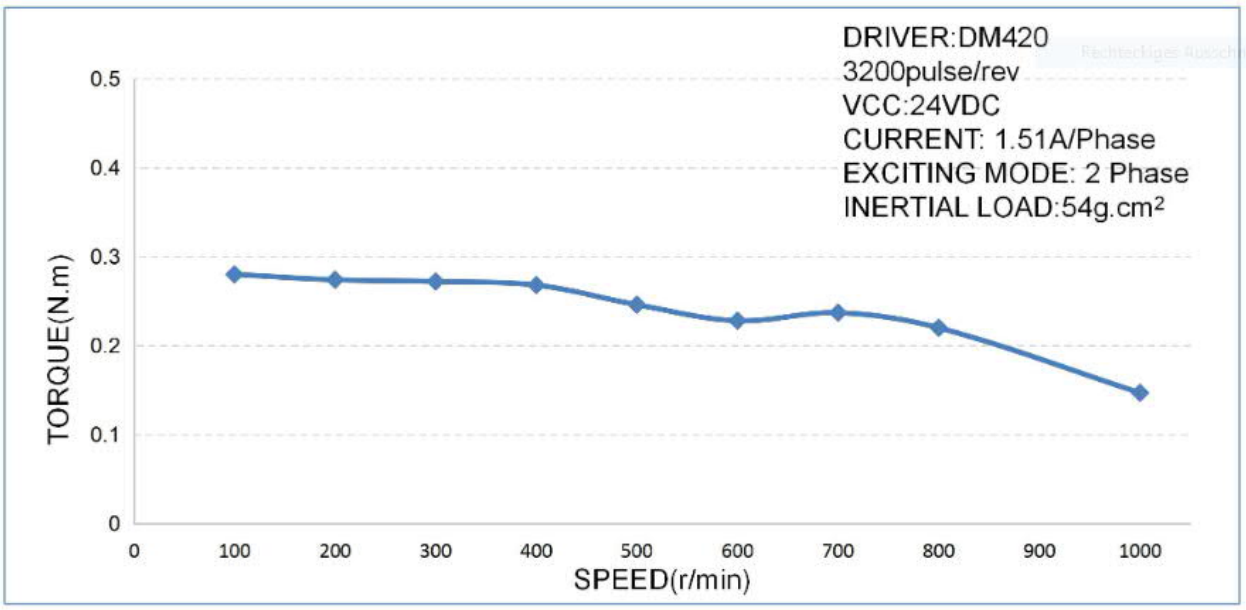


Abbildung 1: Drehmoment über Drehzahl aus einem typischen Datenblatt

[1] Siehe auch: Eichler, H.-D. Kronfeldt, and Sahm (*Das Neue Physikalische Grundpraktikum*) und Translation und Rotation Eichler, Heinz-Detler Kronfeldt, and Sahm

Wenn wir jetzt einen Radius von 100mm annehmen und den Maximalen Strom, dann ergibt sich eine Masse: $m(\alpha) = \frac{0{,}3}{0{,}1^2 \cdot \vec{\alpha}}$, wie unten dargestellt:

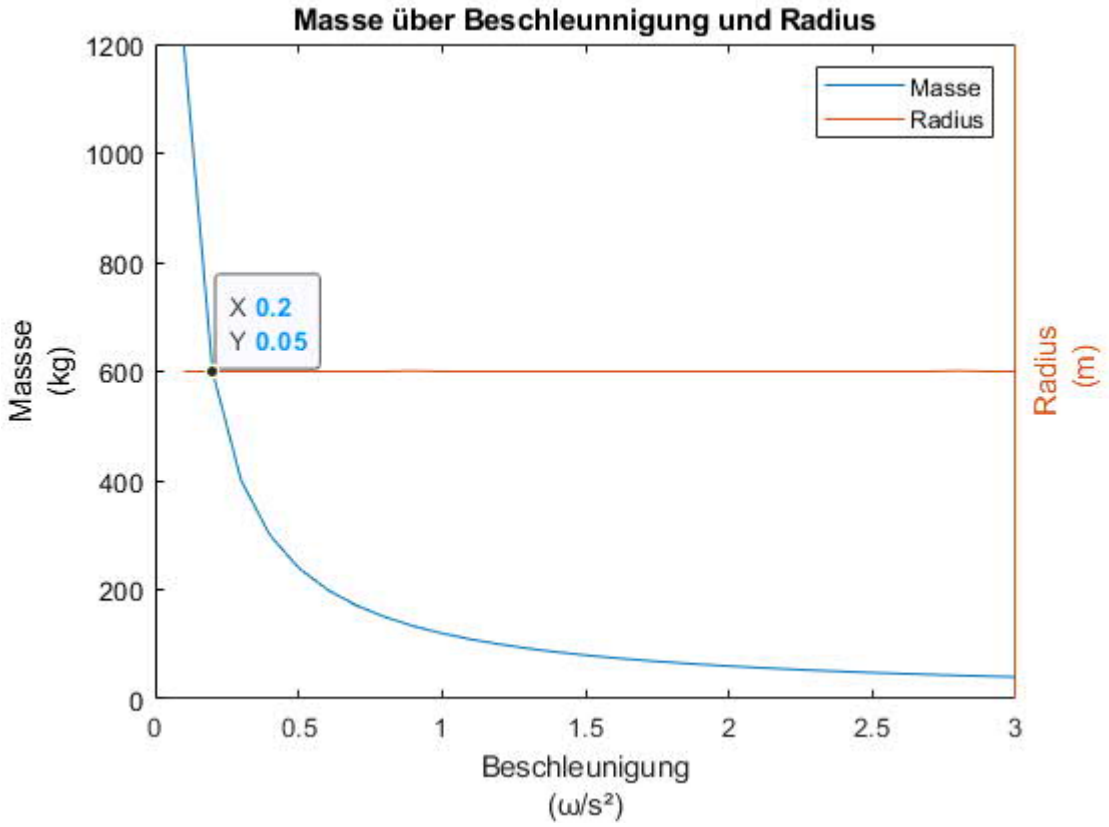


Abbildung 2: Beschleunigung einer Masse mittels eines NEMA17 Schrittmotors

Draus ergibt sich aus den bestehenden Daten eine maximal zu beschleunigendem von 0.6 kg bei Ø 100mm und 0,2 rad/s^2. Es ist also experimentell zu zeigen, ob diese Annahme sich verifiziert.

Serielle Datenkommunikation

Die Datenkommunikation zur Übertragung von Daten zwischen Geräten ist in der Technik essenziell. Diese Daten können binäre Zustände oder analoge, digitale sein. Bei paralleler Kommunikation können zwar mehr Daten gleichzeitig übertragen werden, aber der technische Aufwand und die Fehlerhäufigkeit steigt mit jeder weiteren Verbindung. Deshalb hat sich die serielle Verbindung zwischen Geräten durchgesetzt.

Zwar bekommt Wireless Lan (WiFi, oder IEEE 802.11) und Bluetooth / Low Energy Bluetooth (BLE oder IEEE 802.15.1) immer mehr an Bedeutung, aber der Draht/fasergebundene Kommunikation ist noch immer am ausfallsichersten. Deshalb ist dieses Übertragungsverfahren nicht zuletzt auch im IT-Umfeld so weit verbreitet. (Twisted-Pair-Netzwerkkabelverbindungen im LAN) Im industriellen Umfeld haben sich mehrere Protokolle auf einer seriellen Übertragung etabliert. Währen in den Anfängen der

industriellen Automation EIA232, EIA4xx dominierten, kommen in modernen Anlagen ProfiBus, ProfiNet, EtherCAT u.ä. zur Anwendung. Deshalb werden die in der Technik meistverwendeten Verbindungen jenseits der 90-100m der strukturierten Verkabelung des LANs hier kurz vorgestellt.

EIA232[2]

„RS-232 [..] ist ein Standard für eine serielle Schnittstelle, der in den frühen 1960er Jahren vom US-amerikanischen Standardisierungsgremium Electronic Industries Association (EIA) erarbeitet wurde und bis in die 2010er Jahre häufig bei Computern vorhanden war.[…] Viele aktuelle Geräte mit RS-232 nutzen nur drei Adern bzw. Pins (RX, TX, GND), verzichten also auf die Handshake- und Steuerleitungen. Wegen der niedrigen Datenrate, der geringen Anforderungen an die Verkabelung und des hohen, toleranten Signalpegels ist die RS-232 auch weiterhin verbreitet, wenn es um Störsicherheit und lange Signalverbindungen geht. Sie wird jedoch in dieser Hinsicht von Twisted-Pair-Netzwerkkabelverbindungen mit Transformator („Ethernet“) sowie vom RS-485-Standard übertroffen“. (Wikipedia)

EIA485

EIA485 definiert ein asynchrones Bitübertragungsverfahren (OSI Layer 1) für Direktverbindungen/Punkt-zu-Punkt Verbindungen über eine Datenleitung. Die Datenleitung wird hierbei im Wechselbetrieb (Halbduplex) verwendet, d.h. nur jeweils ein Teilnehmer kann zur gleichen Zeit senden (OSI Layer 2).

Die elektrische Übertragung auf der Datenleitung funktioniert, indem auf der Datenleitung **A**(Y), **B**(Z) nur ein Teilnehmer jeweils senden kann. Alle anderen Teilnehmer lauschen auf demselben Adernpaar. Beide Verbindungsenden müssen mit einem Widerstandswert von typischerweise 120 Ohm terminiert werden, wie in der Abbildung unten dargestellt.

Der Sendertreiber im Baustein (IC) wandelt den EIA232 (TTL)-Pegel 1,8-5V in ein Signal in ein Signal differentieller Polarität (im Gegentakt) um. Dabei ist der Sendertrieber in einem niederohmigen Zustand und der Sendertreiber des Empfängers in einem hochohmigen. Die beiden Leitungen RX(D) und TX(D) des Senderswerden zu den Signalen A(Y) und B(Z) Der Empfänger-SchmittTrigger wandelt die Empfangenen Signale A(Y) und B(Z) wieder zurück in die EIA232 (TTL) Pegel. Danach gehen die Ausgänge des Sender Treibers in einen hochohmigen Zustand über und der Sendertreiber des Empfängers wird in einen niederohmigen zustand überführt. Der Bus ist somit vorbereit, um vom Empfänger eine Antwort zu erhalten. Nach Erhalt der Antwort vom Empfänger startet der Zyklus von neuem.

[2] Früher verwendete man das Präfix RSxxx, die aktuell richtige Präfix EIAxxx hat sich allerdings nicht durchsetzen können. Vgl. hier.

EIA485 kennt die Erweiterung des Datenübertragungsverfahrens auf ein Netzwerk aus mehreren Nodes. Um dies zu ermöglichen, werden die Treiber der Datenleitungen wie oben beschrieben verschaltet. Um Nodes zu adressieren, wird als Netzwerkschicht (OSI Layer 3) eingeführt. Die Organisation des Wechselbetriebs auf OSI Layer 2 erfolgt dabei im Master-Slave Betrieb; nur der vom Master explizit angesprochene Slave darf Daten an den Master zurücksenden; Sonst kommt es zu Kollisionen auf dem Bus.[3]

EIA422

EIA422 definiert ein synchrones Bitübertragungsverfahren (OSI Layer 1) für Mehrfachverbindungen über zwei Datenleitungen. Die Datenleitungen wird hierbei nicht im Wechselbetrieb (Halbduplex) verwendet, sondern für jede Richtung steht eine Datenleitun zu Verfügung. (Volluplex). Im Vierdrahtbetrieb werden die Adernpaare **A**/**B** des Senders mit den Aderpaaren **Y**/**Z** des Empfängers verbunden.[4]

EIA422 definiert auch einen Halbduplex betrieb. Dabei werden die Leitungen **A**/**B** und **Y**/**Z** elektrisch verbunden. Dieser Betrieb benötigt spezielle ICs mit der Fähigkeit die Senderbausteiene hochohmig schalten zu können, wenn auf dem Buss eine Antwort erwartet wird. Siehe Fußnote 3

[3] Folgendes Szenario: Der Leistungstreiber der Leitung A des Bausteins A1 hat den Pegel high. Gleichzeitig hat ein anderer Teilnehmer die Leitung A seines Treibers (A2) auf low Pegel. In der Brückenschaltung der Treiber kommt es jetzt dazu, dass der High-Side Transistor von A1 (durchgesteuert) mit dem Low-Side-Transistor von A2 (durchgesteuert) verbunden wird. Das führt zu einem Kurzschluss und zerstört mindestens einen der beiden Transistoren.

[4] Dadurch, dass die niederohmigen Sender mit den hochohmigen Empfängern verbunden wurden, kann es nicht länger zu Kurzschlüssen auf dem Bus kommen.

EIA422 vs. EIA48

Beide Bezeichnungen werden häufig synonymal verwendet, obwohl es sich bei EIA422 um ein synchrones zwei drahtiges Übertragungsverfahren handelt, während EIA485 ein asynchrones vier drahtiges ist, wie aus den Bildern gut zu erkennen.

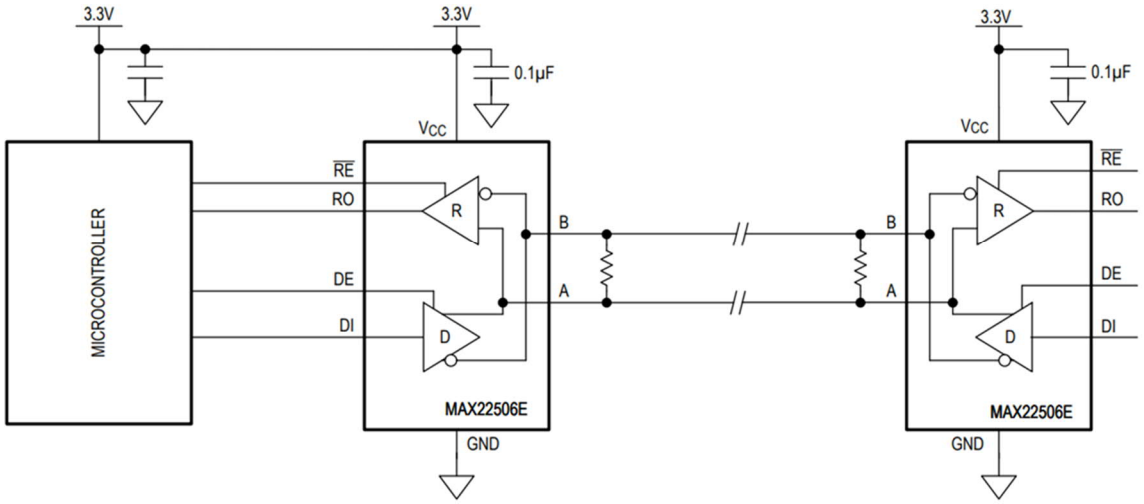


Abbildung 3: EIA485 Verbindung (MAX22506E RS-485-/RS-422-Halbduplex-Transceiver - Maxim | Mouser)

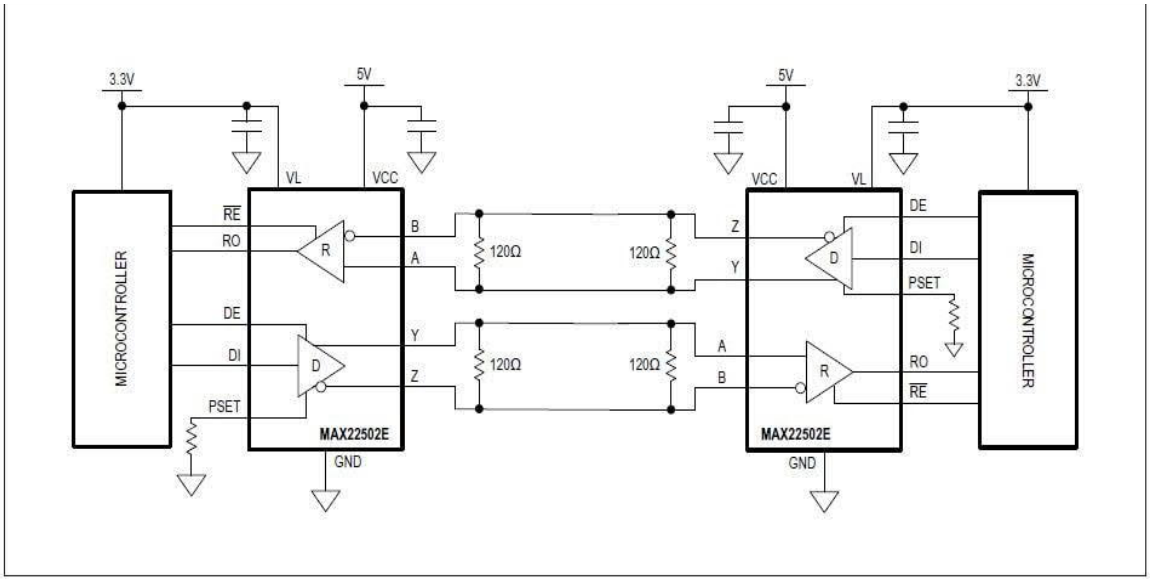


Abbildung 4 EIA422 Verbindung (*MAX22502E RS-485-/RS-422-Transceiver - Maxim | Mouser*)

Folgende Tabelle enthält den wichtigsten technischen Merkmale:

Anschlussname	**RS-422**	**RS-485**
Art der Übertragung	Vollduplex	Halbduplex
Maximale Distanz	1200 Meter bei 9600 bps	1200 Meter bei 9600 bps
Verwendete Kontakte	TxA(**A**), TxB(**Y**), RxA(**B**), RxB(**Z**), GND	DataA, DataB, GND
Topologie	Point-to-Point	Multi-point

Anschlussname	RS-422	RS-485
Max. Anzahl der angeschlossenen Geräte	1 (10 Geräte im Empfangsmodus)	32 (mit größeren Repeatern, normalerweise bis zu 256)

Die effektive, mögliche Übertragungsrate ist von der Kabellänge abhängig. Eine maximale Übertragungsrate von 10 Mbps ist bei einer Kabellänge von ca. 12 m möglich, bei der maximalen Leitungslänge von 1200 m ist nur noch eine maximale Übertragungsrate von etwa 90 kbps möglich. Diese Richtwerte sind, durch geeignete Wahl in der Qualität der Übertragungsleitung und durch die Verwendung besserer Schaltkreise lassen sich diese Werte erheblich verbessern.

Modbus

Modbus definiert mehrere Übertragungsverfahren hier kommt jedoch nur die RTU-Variante zum Einsatz. Modbus funktioniert nach dem Master/Slave Prinzip wie oben beschrieben. Das Protokoll unterstützt bei der Übertragung lediglich binäre, vorzeichenlose 16-Bit Werte, die blockweise vom Master zurück gelesen Das Modbus Protokoll (OSI Layer 3) wurde Anfang der 80er Jahre von der Firma MODICON entwickelt und veröffentlicht. Es wird hauptsächlich im Bereich der Prozessautomatisierung eingesetzt. Da es ein offenes Protokoll ist und eine einfache Struktur besitzt fand es eine weite Verbreitung. Die Verantwortung für die Pflege und Weiterentwicklung des Protokolls liegt mittlerweile bei der Modbus Organisation..[6]

Coils und Diskrete Eingänge sind binär, die anderen beiden digital.

Objekttyp	Zugriff	Größe
Einzelner Ein-/Ausgang „Coil“	Lesen & Schreiben	1-bit
Einzelner Eingang „Discrete Input“	nur Lesen	1-bit
(analoge) Eingänge „Input Register“	nur Lesen	16-bits
(analoge) Ein-/Ausgänge „Holding Register“	Lesen & Schreiben	16-bits

Tabelle 1:Modbus Register

[6] Hier kann auch sämtliche Protokoll-Dokumentation bezogen werden.

Methodik

Als Entwicklungsumgebung wurde PlatformIO und Microsoft VSCode verwendet. Als Hardware kam ein Arduino MEGA 6250 und eine selbst entwickelte Platine zum Einsatz.[10]

Als Bibliotheken wurden verwendet:

Modbus	ModbusSlave
Timer3	TimerThree
Timer4	TimerFour
Ansteuerung Schrittmotortreiber DRV8055	FastAccelStepper
Encoder	RotaryEncoder

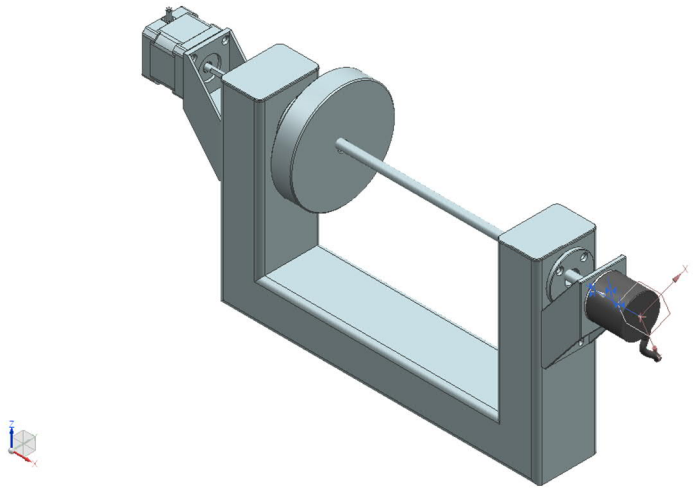
Die Ansteuerung der Hardware und Visualisierung wurde mit Matlab Designer entwickelt. Sie verwendet das Modbus Objekt der Instrument Control Toolbox zur Steuerung der Elektronik. Die Auswertung der Messdaten erfolgt im Polling-Verfahren von der Visualisierung.

Der mechanische Versuchsaufbau ist wie folgt beschrieben:

[10] Die Platine wurde zum jetzigen Zeitpunkt als „Draht-Igel“ gemäß dem im Repository befindlichen Schaltplan erstellt. Es soll dem Auftraggeber die Möglichkeit gegeben werden, daran noch Veränderungen vorzunehmen.

Durchführung

Mechanik



Als Mechanik wurde ein Schweißgestell 60x40x2 (S235-JR) verwendet. Durch zwei Flansche, in denen ein Nadellager verbaut wurden, ist eine geschliffene 8 mm Welle aus C45 gesteckt worden, an deren Enden sich jeweils ein Nema17 Schrittmotor und ein Drehencoder mit einer Auflösung von 400 Schritten befinden. Beide Seiten wurden jeweils durch Kupplungen miteinander verbunden. Des Weiteren ist auf der Welle eine Rotationsmasse aus S235 von ⌀100x14 verbaut.[11]

Elektronik und Firmware

Bei der Programmierung der Firmware wurde auf gute Kommentierung und Verwendung symbolischer Adressen geachtet. Ergänzende Informationen befinden sich im Quellcode. Die Verarbeitung der Modbus-Daten erfolgt im polling Verfahren. Nachdem ein Datum gelesen wurde, wird kontrolliert die Bibliothek ModbusSlave, ob jenes einen in der Firmware registrierten MB-Befehl zugeordnet werden kann. Im Erfolgsfall wird ein registriertes Callback ausgelöst welches das übermittelte Register, auf die die u. g. globalen Variablen überträgt.

```
Volatile static bool gCoils[30]={}
volatile static bool gContacts[30]={}
volatile static uint16_t gInputs[30]={}
volatile static uint16_t gHoldings[30]={}
```

Diese Variablen werden in der Hauptschleife der Arduino-Frameworks weiterverarbeitet:

[11] Datenblätter und Zeichnungen befinden sich im Repository.

```
ine gSamplingRate      gHolding[3]   // samplingrate of measurement [Hz]

  // start or stopsa messurement on the App
  if ( gStartMeassure ){
      delay(10); // fills the gap
      if (millis() > gFinishTime){
        gMessureIsRunning = false;
        Timer3.stop();
        gStartMeassure=false;
      } else {
        Timer3.setPeriod( (unsigned long)(1.0E6/gSamplingRate) );
        gMessureIsRunning = true;
      }
  } else {
      gStartTime  = millis();
      gFinishTime = millis() + gMessuringDuration;
  }
```

Hier wird das weiter unten beschriebene Problem der asynchronen Verarbeitung deutlich:

Das coil Register *gStartMeassure* triggert eine neue Messung. Die Samplingrate wird fortlaufend aktualisiert und in den Timer4 der MCU übertragen. Der Callback dieses Timers überträgt dann bei Überlauf ein Datum auf die serielle Schnittstelle. Das *discrete imput gMessureIsRunning* wird so lange aktualisiert, bis der System-Zeitstempel 10 ms größer als *discrete imput* gFinishTime. Danach wird der Timmer 4 wieder gestoppt und es werden keine weiteren Daten übertragen.

Findet keine Messung statt, werden die Input Register *gStartTime* und *gFinishTime* mit Zeitstempel gefüllt. Damit die Visualisierung das Ende der Messung mitbekommt, muss sie das Register *gMessureIsRunning* auslesen. Dazu pollt sie die benötigten Stati von der Firmware.

Der *tmTimer3* erzeugt über den dazugehörigen *cbWriteMessurementData*. Der Üerlaufwert des Timers in ms entspricht der Samplingrate für die Messwerte.

Über die präprozessordefinition IS_WITHOUT_HARDWARE wird gesteuert, ob Testaten übermittelt werden. In der Entwicklung ohne Hardware werden Werte aus sin(*gRadTimeStamp*) , cos(*gRadTimeStamp*) und tan(*gRadTimeStamp*) generiert werden und diese an die Datenschnittstelle geschrieben.

Die Realen Messdatenwerden werden über den aktuellen Wert von *gCurrentEncoderPostition* , *gCurrentEncoderAngle* und *gCurrentStepperValue* an die Daten- Schnittstelle geschrieben.

Die Werte des Encoder wird mittels Interrupts der Pins zu den Signale A und B des Encoders generiert und in *gCurrentEncoderPostition* und *gCurrentEncoderAngle* abgelegt.

Die Funktion der Anstuerung des Schrittmotors ist in seiner API gut beschrieben.

Der *tmTimer4* erzeugt über den dazugehörigen *cbMsTick* Callback ein Zeitnormal von 1 ms. In der Entwicklung ohne Hardware wird hier der der Wert für die Funktionen sin(x) und cos(x) erzeugt und in *gRadTimeStamp* abgelegt.

Matlab App

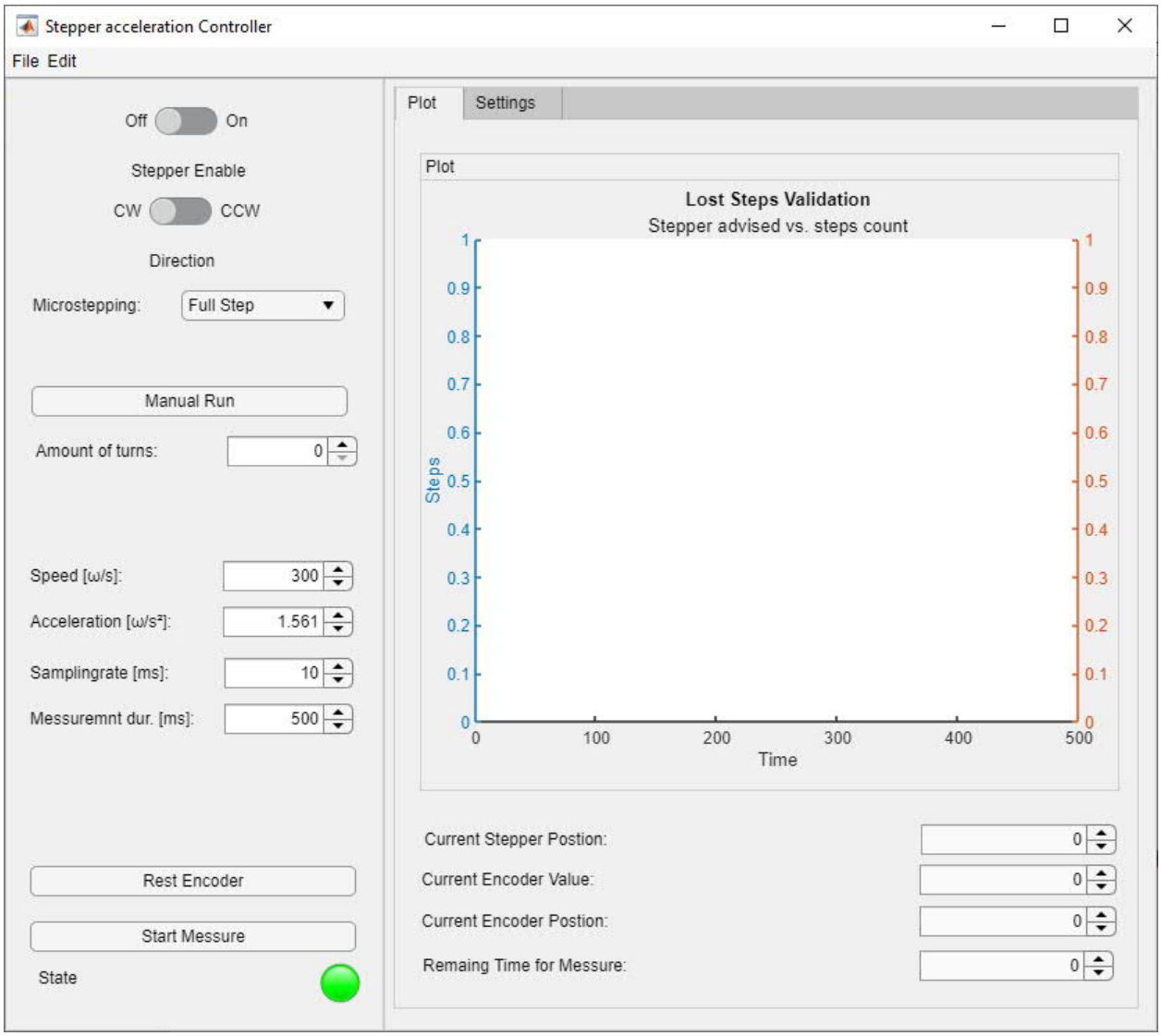


Abbildung 5: Oberfläche der Matlab Applikation

Die Anwendung ermöglicht es die oben beschriebene Mechanik ober die oben beschriebene Elektronik zu visualisieren. Sie ist mit den Mechanismen der asyncronen Datenverarbeitung umgesetzt.

Analog zur Firmware werden in der App asynchron über Events Callbacks ausgeführt. Das Ereignis *LineFeedReceived (0xA)* triggert den Eventhandler *cbReadSerialData(app,src,~).* Dieser extrahiert die Daten aus dem Stream und hängt diese an die Struktur *stk*S*treamingData* an. Schließlich werden die Daten aus der Struktur *skStreamingData*.stepperSteps und *skStreamingData.*ecoderSteps über *skStreamingData*.timestamp geplottet.

Die zweite asynchrone Verarbeitung erfolgt über den Timer *tmModbusTimer* und *cbReadSerialData* als dazugehöriges Callback. Hier werden periodisch die Modbus-Register gelesen. Idealerweise sollte dazu die Modbus Funktion (0x17, Read/Write Multiple register) verwendet werden, das wird aber von der Firmware (Modbus-Bibliothek) nicht unterstützt.

Alle Widgets der Anwendung haben eigene, vorgegebene Ereignisse, die ihre dazugehörigen Callbacks triggern können. Hier wurde zunächst versucht das Schreiben der Modbus Register zu implementieren. Die Modbus Implementierung von Matlab ist statuslos und nicht blockierend. Das führt dazu, dass auf der Schnittstelle parallel mehrere Zugriffe ausgeführt werden. Dies führt zu CRC-Fehlern im Objekt *obModBusInterface* welches diesen Fehler an den Timer *tmModbusTimer* weiter reicht und diesen dann stoppt. Somit findet kein weiterer Pollen mehr statt und die Anwendung muss neu gestartet werden. Lösungsvorschläge werden weiter unten diskutiert.

Diskussion

Firmwareseitig ist das Senden von CSV-Daten benutzerfreundlich. Da der Offset deutlich höher liegt ist gerade bei einer so schwachen MCU wie den AtMega6250 das binäre Senden sinnvoller. Auch ist die Übertragungsgeschwindigkeit von 0,5Mbit sehr hoch gewählt und ist der Entscheidung das Projekt im CSV-Format umzusetzen geschuldet. Bei einer binären Übertragung und einem Datenrahmen von 8N1 ergeben sich 288 Bits für ein Datum. Bei CSV sind das neun Bits für je Zeichen. Da es sich in dieser Arbeit vor allem um ein Proof of Concept handelt, ist die Implementierung dessen in einem Release zu überlegen.

Die Wahl der Ansteuerung mittels Modbus erwies sich als problematisch. Als Lösung könnte überlegt werden, ob die Implementierung eines eigenen Ereignissystems als Instanz der Klasse *matlab.DiscreteEventSystem* sinnvoll wäre, aber es würde den Rahmen dieser Arbeit sprengen.

Ein anderer Ansatz wäre es in die MB-Firmware den MB-Befehl readwrite (0x17) zu implementieren. Dann erfolgten weniger Zugriffe auf den modbus Port.

Ein letzter Ansatz wäre das Lesen/schreiben der MB-Register mit einem try-catch abzufangen, was aber erst zum Abgabezeitpunkt als Lösungsmöglichkeit in Betracht gezogen worden ist.

Möglicherweise wäre die Verwendung von z. B. SCPI aus der „Instrument Control Toolbox“ von Matlab der beste Weg, da es sich ja hier nicht um eine Steuerung einer Anlage handelt, sondern eher um ein Messinstrument.

Anhang:

Alle Datenblätter, Quellcodes und Zeichnungen sind hier zu finden. Aktuelle Versionen ab dem 01.03.2022 sind in der Historie einsehbar.

Glossar

MCU	**M**ikro**c**ontroller **U**nit (auch *µController*, *µC*, *MCU*)

MB	**M**od**B**us
bps	**B**its **P**er **S**econd, Norm einer seriellen Datenübertragungseschwindigkeit. Früher auch Baud genannt
TTL	**T**ransistor-**T**ransistor-**L**ogik
IC	**I**ntegrated **C**ircuit
LAN	**L**ocal **A**rea **N**etwork, Netzwerksystem einer Organisation innerhalb dessen lokalem Gelände.

Callback -Prinzip

„Eine Rückruffunktion (englisch Callback) bezeichnet in der Informatik eine Funktion, die einer anderen Funktion, meist einer vorgefertigten Bibliotheks- oder Betriebssystemfunktion, als Parameter übergeben und von dieser unter definierten Bedingungen mit definierten Argumenten aufgerufen wird. Dieses Vorgehen folgt dem Entwurfsmuster der Inversion of Control.

Meistens erlaubt die vorgefertigte Funktion die Übergabe eines sog. Benutzerparameters app, event, der von ihr (neben anderen Argumenten) zur Rückruffunktion durchgereicht wird, damit letztere im Kontext des ursprünglichen Aufrufers Daten sowohl abgreifen wie ablegen kann.
Es gibt zwei Arten von Rückrufen, die sich darin unterscheiden, wie sie den Datenfluss zur Laufzeit steuern: blockierende Rückrufe (auch bekannt als synchrone Rückrufe oder einfach Rückrufe) und zeitversetzte Rückrufe (auch bekannt als asynchrone Rückrufe). Während blockierende Rückrufe vor der Rückkehr einer Funktion aufgerufen werden, können zeitversetzte Rückrufe nach der Rückkehr einer Funktion aufgerufen werden. Aufgeschobene Rückrufe werden häufig im Zusammenhang mit E/A-Operationen oder der Behandlung von Ereignissen verwendet und werden durch Unterbrechungen oder im Falle mehrerer Threads durch einen anderen Thread aufgerufen. Aufgrund ihrer Beschaffenheit können blockierende Rückrufe ohne Unterbrechungen oder mehrere Threads funktionieren, was bedeutet, dass blockierende Rückrufe in der Regel nicht für die Synchronisierung oder das Delegieren von Arbeit an einen anderen Thread verwendet werden“. (Rückruffunktion – Wikipedia)

Bildverzeichnis

Modbus Register

0x01	Read Coils
Ox02	Read Discrete Input

Ox03	Read Holding Register
0x04	Read Input Registers
0x05	Write Single Coil
0x06	Write Single Register
(0x0F)	Write Multiple Coils
(0x10)	Write Multiple registers
~~(0x17)~~	~~Read/Write Multiple registers~~[16]

Mosbus Symbolik

Coils

```
MB_COIL_STEPPER_ENA             =     1      % stepper direction 0=CW; 1=CCW
MB_COIL_STEPPER_DIR_CCW         =     2      % Enables the stepper
MB_COIL_STEPPER_RUN             =     3      % Run the stepper
MB_COIL_STEPPER_MOVE            =     4      % Run the stepper clockwise
MB_COIL_START_MEASSURE          =     5      % Start one measurement cycle
MB_COIL_ZERO_ENCODER            =     6      % sets the encoder to zero
```

[16] Leider ist der Befehl 0x17 noch nicht implementiert. Das führt in dem oben beschriebenen Zusammenhang dazu, dass nur mit erheblichem Zeitaufwand das modbus Protokoll sich asynchron mit der Matlab-Implementation verwenden lässt.

Contacts (discrete inputs)

```
MB_CONTACT_STEPPER_IS_RUNNING     =   1      % indicates weather the stepper is
running
MB_CONTACT_STEPPER_IS_STOPPING    =   2      % indicates weather the stepper is
stopping
MB_CONTACT_MESSUREMENT_IS_RUNNING =   3      % indicates weather the stepper is
stopping
```

Inputs

```
MB_INPUT_SYSTEM_TIME              =   1      % current system Time (LSB)
MB_INPUT_SYSTEM_TIME_START        =   5      % current system Time at start (LSB)
MB_INPUT_SYSTEM_TIME_FINISH       =   9      % current system Time at finish
(LSB)
MB_INPUT_STEPPER_STEPS            =  13      % Current steps from the stepper
driver (LSB)
MB_INPUT_ENCODER_STEPS            =  17      % current steps from the encoder
MB_INPUT_ENCODER_ANGLE            =  21      % current (calculated) Angle from
the encoder (rad*1000)
```

Holding registers

```
MB_HOLDING_STEPPER_TURN_STEPS     =   1      % stepper amount  of steps per turn
MB_HOLDING_ENCODER_TURN_STEPS     =   2      % encoder amount of steps per turn
MB_HOLDING_MESSUREMENT_DURATION   =   3      % length of measurement [ms]
MB_HOLDING_SAMPLE_RATE            =   4      % samplingrate of measurement [Hz]
MB_HOLDING_STEPER_MICROSTEPS      =   5      % Micosteps from 2^0..2^5
MB_HOLDING_STEPPER_ACCEL          =   6      % turning acceleration rad*1000
[ω/s²]
MB_HOLDING_STEPPER_SPEED          =   7      % turning speed [ω/s²]
MB_HOLDING_STEPPER_TURNS_TO_RUN   =   8      % amount of steps to run 0=inf
```

Serielle Parameter

Modbus

Übertragungsrate:	9600
Datenbits	8
Parität:	none
Stopbits	1

Daten

Übertragungsrate:	500000
Datenbits	8

Parität:	none
Stopbits	1

Debugging

Übertragungsrate:	115200
Datenbits	8
Parität:	none
Stopbits	1

5. Works Cited

Eichler, Hans J., Heinz-Detlef Kronfeldt, and Jürgen Sahm, eds. *Das Neue Physikalische Grundpraktikum*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2001. Print. Springer eBook Collection Life Science and Basic Disciplines.

Eichler, Hans J., Heinz-Detler Kronfeldt, and Jürgen Sahm. "Translation und Rotation." *Das Neue Physikalische Grundpraktikum*. Ed. Hans J. Eichler, Heinz-Detlef Kronfeldt, and Jürgen Sahm. Berlin, Heidelberg, s.l. Springer Berlin Heidelberg, 2001. 33-42. Springer eBook Collection Life Science and Basic Disciplines. Print.

MAX22502E RS-485-/RS-422-Transceiver - Maxim | Mouser, 2022, 25 Feb. 2022. Web. 25 Feb. 2022. <https://www.mouser.de/new/maxim-integrated/maxim-max22502e-transceiver/>.

MAX22506E RS-485-/RS-422-Halbduplex-Transceiver - Maxim | Mouser, 2022, 25 Feb. 2022. Web. 25 Feb. 2022. <https://www.mouser.de/new/maxim-integrated/maxim-max22506e-half-duplex-transceiver/>.

Rückruffunktion – Wikipedia, 2022, 13 Feb. 2022. Web. 27 Feb. 2022. <https://de.wikipedia.org/wiki/R%C3%BCckruffunktion>.

Wikipedia, ed. *RS-232*, 2021, 12 Nov. 2021. Web. 28 Feb. 2022. <https://de.wikipedia.org/w/index.php?title=RS-232&oldid=217204920>.