Ein Bild, das Text enthält.

Automatisch generierte BeschreibungJörg Bernau (15411098)

Theodor Kiesel

## Informatik imMaschinenbau (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. Er beinhaltet die Erstellung der Firmware und einer Matlab App zur Steuerung des Versuchs.

Inhalt

[Informatik imMaschinenbau (VIM2) 1](#_Toc96862579)

[Theorie 2](#_Toc96862580)

[Widerstands- und Drehmomentmoment 2](#_Toc96862581)

[Serielle Datenkommunikation im Allgemeinen 3](#_Toc96862582)

[RS 422 vs. RS485 3](#_Toc96862583)

[EIA485 5](#_Toc96862584)

[EIA422 6](#_Toc96862585)

[Modbus 6](#_Toc96862586)

[Matlab App 7](#_Toc96862587)

[Allgemines 7](#_Toc96862588)

[Callback-Prinzip 8](#_Toc96862589)

[Konzept der App 8](#_Toc96862590)

[Elektronik-Hardware und Firmware 9](#_Toc96862591)

[Allgemeines 9](#_Toc96862592)

[Die Parameter der seriellen Kommunikation 10](#_Toc96862593)

[Modbus Register 11](#_Toc96862594)

[Coils 11](#_Toc96862595)

[Mechanik 12](#_Toc96862596)

[Glossar 12](#_Toc96862597)

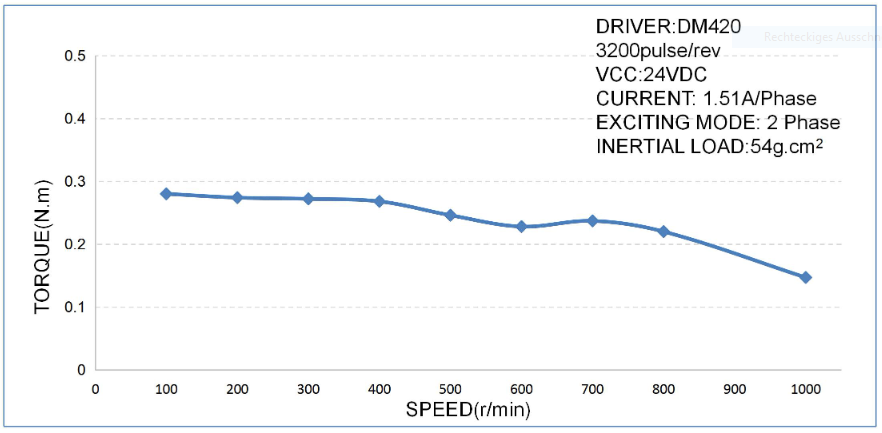
[Anhang: 14](#_Toc96862598)

[5. Quellen und Literaturverzeichnis 14](#_Toc96862599)

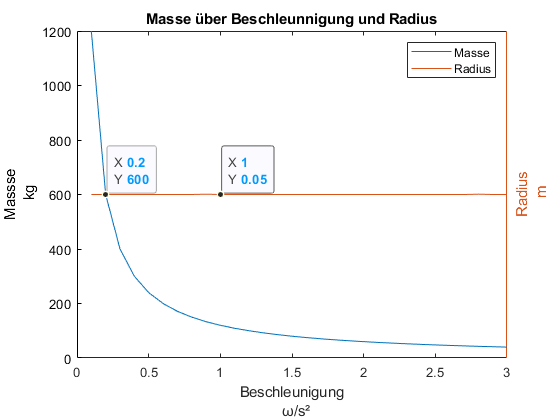
# Theorie

## Widerstands- und Drehmomentmoment

Für die Translation gilt zwischen der Kraft , der Masse m und der Beschleunigung der grundlegende Zusammenhang das newtonsche Grundgesetz. Für die Rotation starrer Körper gilt analog das Grundgesetz der Dynamik der Rotation: . Wird mit substituieret ergibt sich



Wenn wir jetzt einen Radius von 100mm annehmen und den Maximalen Strom, dann ergibt sich eine Masse: *,* wie unten dargestellt:



Draus ergibt sich aus den bestehenden Daten eine maximal zu beschleunigendem von 0.6 kg bei Ø 100mm und 0,2 rad/s2.

## Serielle Datenkommunikation

Die serielle Datenkommunikation ist in der Technik essenziell. 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. (ugs. Ethernet) Im industriellen Umfeld haben sich mehrere Protokolle auf einer seriellen Übertragung etabliert. Währen in den Anfängen der industriellen Automation EIA232, EIA4xx dominierten, sind in modernen Anlagen die Begriffe ProfiBus, ProfiNet, EtherCAT u.ä. zu finden.

## RS 422 vs. RS485

Die Verwendung der Kennung RSxxx ist historisch, aber noch sehr verbreitet. Weniger bekannt ist die neue Bezeichnung EIAxxx Beide Bezeichnungen werden synonymal verwendet, obwohl es sich bei EIA422 um ein synchrones Übertragungsverfahren handelt, während EIA485 ein asynchrones ist. Folgende Tabelle enthält den wichtigsten technischen Merkmalen:

| **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 |
| **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

Der hervorstechendste Unterschied ist in der Verdrahtung bildlich dargestellt:

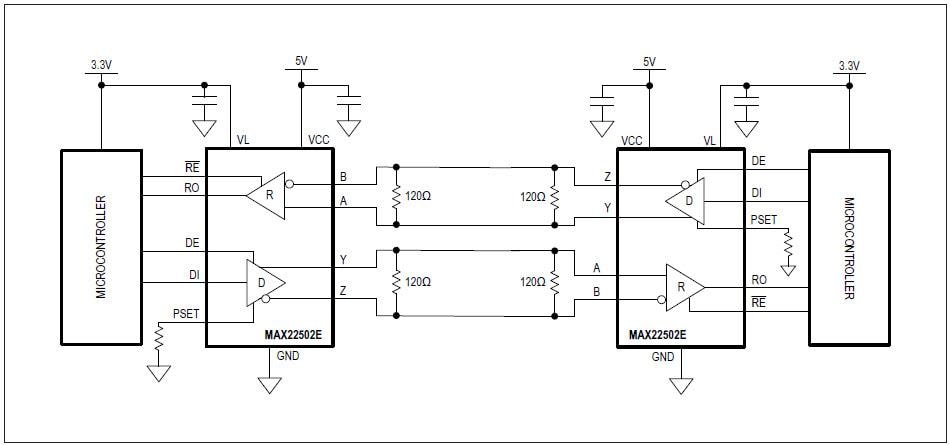


Abbildung 1 EIA422 Verbindung [(MAX22502E RS-485-/RS-422-Transceiver - Maxim | Mouser)](#_CTVL001fdde56a62ab5494a90b53f5626e3624c)

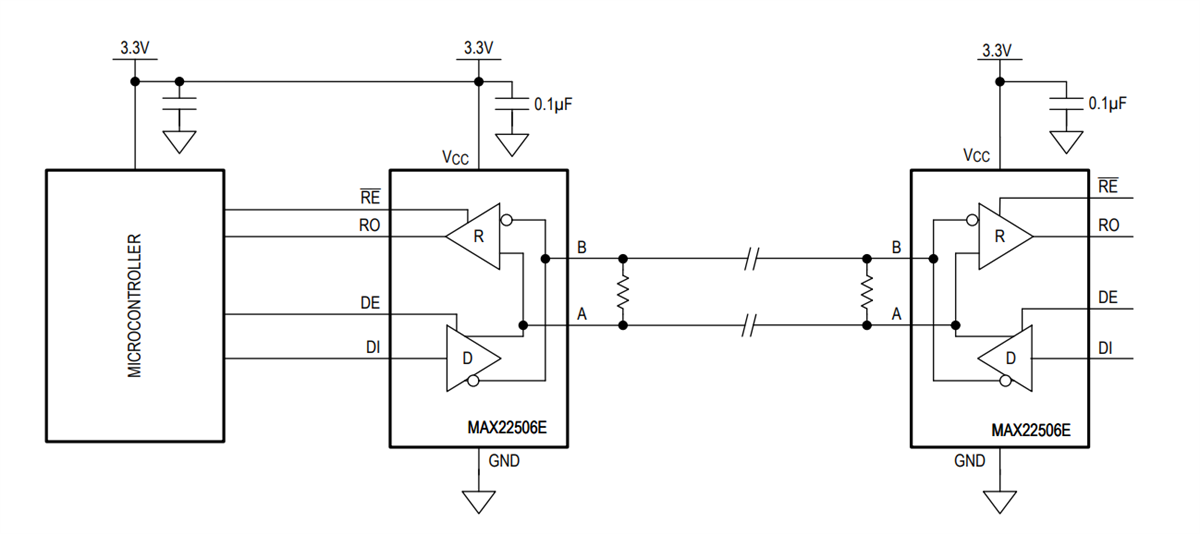


Abbildung 2 EIA485 Verbindung [(MAX22506E RS-485-/RS-422-Halbduplex-Transceiver - Maxim | Mouser)](#_CTVL001729685d0534f4b888a6bf5741da2109e)

Beide Verbindungen müssen wie gezeichnet mit einem typischen Widerstand von 120 Ohm terminiert werden, wie in den Bildern beschrieben.

### EIA485

EIA485 ist ein asynchrones Übertragungsverfahren im Halbduplex Modus, bei dem auf der Datenleitung **A**(Y), **B**(Z) nur ein Teilnehmer jeweils senden kann und ein Teilnehmer empfangen kann. Neben dieser Punkt-zu-Punkt Übertragung hat sich auch eine Master-Slave Übertragung etabliert.

Ein sog. Master sendet deshalb an ein oder mehrere Teilnehmer im Netzwerk. Damit diese Teilnehmer sich voneinander unterscheiden können, bedarf es einer weiteren Schicht auf Layer 2 im OSI Datenmodell. Ein Master kann mit einem oder mehreren Slaves kommunizieren. Nur der vom Master explizit angesprochene Slave darf Daten an den Master zurücksenden; Sonst kommt es zu Kollisionen auf dem Bus, welche nicht erlaubt sind.

Es bedarf einer weiteren Schicht auf Layer 2 im OSI Datenmodel, dessen Protokoll dann die vom Master adressierten Slaves voneinander unterscheiden.

Der Sendertreiber im Baustein (IC) wandelt den EIA232 (TTL)-Pegel 1,8-5V in ein Signal in ein Signal differentieller Polarität im Gegentakt um. Die beiden Leitungen RX(D) und TX(D) werden 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

### EIA422

EIA422 ist ein synchrones Übertragungsverfahren im Vollduplex Modus, bei dem auf der Datenleitung **A**/**B** nur ein Teilnehmer jeweils senden kann und mehrere Teilnehmer auf den Leitungen **Y**/**Z** empfangen können. Ein sog. Master sendet deshalb, wie oben beschrieben, an ein oder mehrere Teilnehmer an den Bus. EIA422 kann auch im Halbduplex betrieben werden.

Auch hier bedarf es auch hier einer weiteren Schicht auf Layer wie oben beschrieben.

Der Sendertreiber im Baustein (IC) wandelt den EIA232 (TTL)-Pegel 1,8-5V in ein Signal in ein Signal differentieller Polarität im Gegentakt um. Die beiden Leitungen RX(D) und TX(D) werden zu den Signalen **A**(Y) und **B**(Z) Der Empfänger-SchmittTrigger wandelt die Empfangenen Signale **Y** und **Z** wieder zurück in die EIA232 (TTL) Pegel

### Modbus

Das Modbus Protokoll wurde Anfang der 80er von der Firma MODICON entwickelt und veröffentlicht. Es wird hauptsächlich im Bereich der Prozessautomatisierung eingesetzt. Da es ein offenes Protokoll ist und nicht zuletzt durch seine einfache Struktur fand es eine weite Verbreitung. Die Verantwortung für die Pflege und Weiterentwicklung des Protokolls liegt mittlerweile bei der [Modbus Organisation](https://modbus.org/). Hier kann auch sämtliche Dokumentation bezogen werden.

Modbus definiert zwei verschiedene Übertragungsverfahren:

* Modbus seriell  
  unterstützt die Kommunikation über serielle Schnittstellen wie RS232, RS485
* Modbus TCP/IP  
  unterstützt die Kommunikation über ein TCP/IP Netzwerk

Für Modbus seriell werden zwei verschieden Übertragungsmodis definiert:

* **Modbus RTU[[1]](#footnote-2)**  
  kodiert die Daten binär
* Modbus ASCII  
  kodiert die Daten mittels ASCII-Zeichensatz in Form von lesbaren Zeichenketten

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

Diese Tabelle verdeutlich das geschriebene:

|  |  |  |
| --- | --- | --- |
| **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 |

Die weitere Spezifikation ist bei der [Modbus Organisation](https://modbus.org/) zu [hier](https://modbus.org/specs.php) finden.

# Matlab App

## Allgemines

Die App wurde mit Matlab Designer entwickelt. Sie verwendet das Modbus Objekt, das zu dem Instrument Control Toolbox gehört.

## 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)](#_CTVL00132d1a7ef473f4c9fa64736ff3a735a6f)

## Konzept der App

Analog zur Firmware wird in der App wird asyncron über events callbacks ausgeführt. Das Ereignis *LineFeedReceived (0xA)* triggert den eventhandler *cbReadSerialData(app,src,~).* Deser extrahiert die Daten aus dem stream und fügt sie in der Struktur *streamingData* hinzu. Schließlich werden *streamingData*.stepperSteps und *streamingData.*ecoderSteps über *streamingData*.timestamp geplottet.

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

Alle Widgets der Anwendung haben eigene Ereignisse, die weitere callbacks triggern können. Hier wurde zunächst versucht das Schreiben der Modbus Register zu implementieren. Da die Modbus Implementierung von Matlab ist leider statuslos und nicht blockierend. Das führt dazu, dass auf der Schnittstelle parallel ausgeführte Zugriffe ausgeführt werden. Das callbacks des Timers *modbusTimer* bekommt einen Fehler und der Timer stopt. Somit findet kein weiteres pollen mehr statt und die Anwendung muss neu gestartet werden.

Eine eigene Implementierung eines eigenen Ereignisystems als Instanz der Klasse *matlab.DiscreteEventSystem* würde den Rahmen der (unbenoteten) Hausarbeit sprengen. Im rahmen der persönlichen Weiterbildung erscheint mir der Weg über die „Instrument Control Toolbox“ von matlab ein besserer weg zu sein.

# Elektronik-Hardware und Firmware

Allgemeines

Als Entwicklungsumgebung wurde [PlatformIO](https://platformio.org/) und [Microsoft VSCode](https://code.visualstudio.com/) verwendet. Als Hardware kam ein [Arduino MEGA 6250](http://store.arduino.cc/products/arduino-mega-2560-rev3) und eine selbst entwickelte [Platine](https://github.com/Beerlesklopfer/VIM2-AccelControler/tree/main/Zeichnungen/Autodesk%20Eagle) zum Einsatz.[[2]](#footnote-3)

Bei der Programmierung der Firmware wurde auf gute Kommentierung und Verwendung symbolischer Adressen geachtet. Weitere Informationen befinden sich im Quellcode. Die Fremdbibliothek [Arduino Modbus Slave](https://github.com/yaacov/ArduinoModbusSlave?utm_source=platformio&utm_medium=piohome) verarbeitet die Befehlskommunikation. Diese Bibliothek implementiert folgende Befehle:

|  |  |
| --- | --- |
|  | Read Coils |
|  | Read Discrete Input |
|  | Read Holding Register |
|  | Read Input Registers |
|  | Write Single Coil |
|  | Write Single Register |
| (0x0F) | Write Multiple Coils |
| (0x0F) | Write Multiple registers |
| ~~(0x17)~~ | ~~Read/Write Multiple registers[[3]](#footnote-4)~~ |

[FastAccelStepper](https://github.com/gin66/FastAccelStepper?utm_source=platformio&utm_medium=piohome)

Die Parameter der seriellen Kommunikation

#define MODBUS\_BAUDRATE    115200       // modbus baudrate

#define STREAMING\_BAUDRATE 500000       // streaming baudrate

#define DEBUGGING\_BAUDRATE 115200 // debugging baudrate

Als Datenrahmen wurde sämtlich 8 Datenbits, keine Parität und 1 Stoppbit (8N1) verwendet.

Das Konzept der Modbus Implementierung ist, dass die bekannten vier Modbus-Register als vier globale Arrays abgebildet werden:

volatile static bool gCoils[30]={}

volatile static bool gContacts[30]={}

volatile static uint16\_t gInputs[30]={}

volatile static uint16\_t gHoldings[30]={}

In der loop Methode des Arduino Frameworks werden diese Variablen Aktionen zugeordnet.

#define 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;

  }

In diesem Anschnitt wird das Problem der asynchronen Bearbeitung 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* wirdso lange aktualisiert, bis der System-Zeitstempel 10 ms größer als *discrete imput* gFinishTime. Danach wird der Teimer 4 wieder gestoppt und es werden keine weiteren daten übertragen.

Be keiner Messung werden die Input Register *gStartTime* und *gFinishTime* mit Zeitstempel gefüllt.

Damit die Matlab-App das Ende der Messung mitbekommt, muss sie das Register *gMessureIsRunning* auslesen. Deshalb verwendet sie einen asynchronen Timer. Das führt zu den oben beschriebenen Problemen.

## Modbus Register

Folgende Modbus-Register finden Verwendung:

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

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

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

MB\_INPUT\_ENCODER\_STEPS            =    17      % current steps from the encoder

MB\_INPUT\_ENCODER\_ANGLE            =    21      % current (calculated) Angle from the encoder [rad\*10000]

MB\_HOLDING\_STEPPER\_TURN\_STEPS     =     1      % stepper: amout of steps per turn

MB\_HOLDING\_ENCODER\_TURN\_STEPS     =     2      % encoder: amout of steps per turn

MB\_HOLDING\_MESSUREMENT\_DURATION   =     3      % length of measurement [ms]

MB\_HOLDING\_SAMPLE\_RATE            =     4      % samplingrate of measurement [Hz]

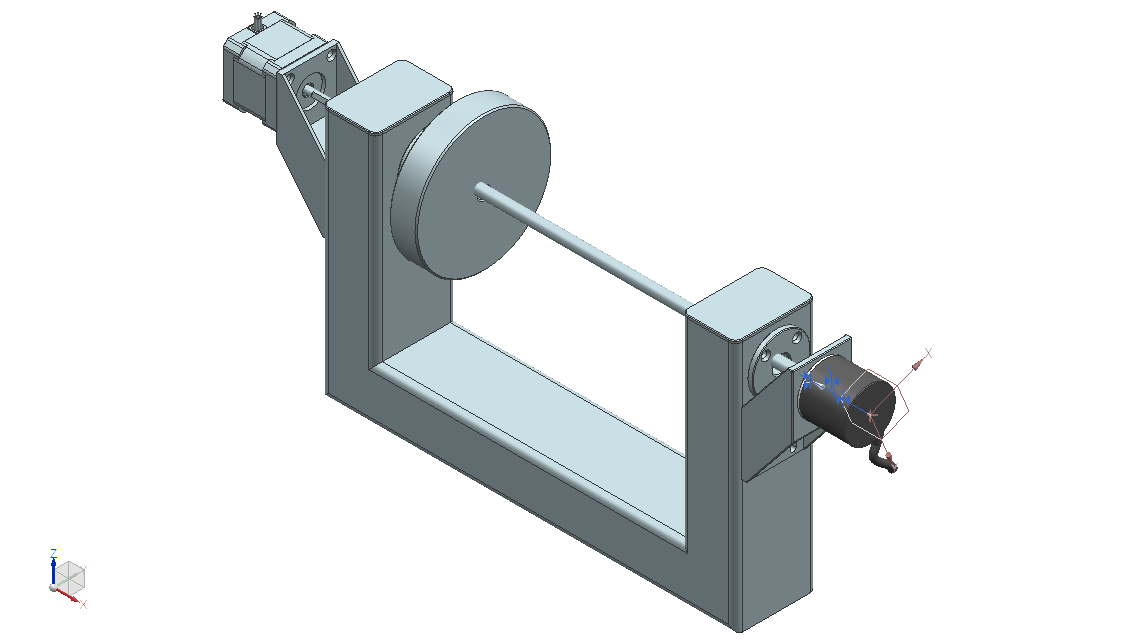
MB\_HOLDING\_STEPER\_MICROSTEPS      =     5      % Micosteps from 2^0..2^4

MB\_HOLDING\_STEPPER\_ACCEL          =     6      % turning acceleration [ω/s²]

MB\_HOLDING\_STEPPER\_SPEED          =     7      % turning speed [ω/s²]

MB\_HOLDING\_STEPPER\_TURNS\_TO\_RUN   =     8      % amount ofsteps to run 0=inf

# Mechanik



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

# Glossar

|  |  |
| --- | --- |
| **MCU** | **M**ikro**c**ontroller **U**nit (auch *µController*, *µC*, *MCU*) |
| **bps** | **B**its **P**er **S**econd, Norm einer seriellen datenbertragungsgeschwindigkeit. Frügerauch Baud genannt |
| **TTL** | **T**ransistor-**T**ransistor-**L**ogik |
| IC | **I**ntegrated **C**ircuit |

# Anhang:

Alle Datenblätter, Quellcodes und Zeichnungen sind [hier zu finden](https://github.com/Beerlesklopfer/VIM2-AccelControler). Aktuelle Versionen ab dem 01.03.2022 sind in der Historie einsehbar.

1. Quellen und Literaturverzeichnis

*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>.

1. Aktuell wird praktisch nur das Modbus RTU Verfahren verwendet. [↑](#footnote-ref-2)
2. 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. [↑](#footnote-ref-3)
3. 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. [↑](#footnote-ref-4)