Einrichtung und Verwendung der Drehscheibe

Kalibrierung der Drehscheibe

Die Drehscheibe wird sich nur im Idealfall spielfrei und völlig gleichmäßig ohne zu ruckeln drehen. Gerade bei Selbstbau-Drehscheiben, wie meiner, werden bei einer Umdrehung und bei Richtungswechsel durch Abbremsen und Beschleunigen sowie unterschiedliche Reibung Microsteps verloren gehen. Die Steuerung der Drehscheibe ist so ausgelegt, dass diese Ungenauigkeiten in einem großen Bereich ausgeglichen werden können.

Damit der Stepper aber keine oder möglichst wenige dieser Microsteps „verliert“, sollten die Einstellungen der Werte für den Stepper den Vorgaben der Datenblätter entsprechen und sorgfältig eingestellt werden. Die in den Datenblättern angegebenen Versorgungsspannungen sind für uns nicht wichtig. Diese bezieht sich auf den Betrieb mit einer einfachen nicht Strom geregelten Stepper-Platine. Bei der A4988-Platine und allen anderen wird der Strom automatisch geregelt. Er wird über das Poti auf der Platine eingestellt indem die Spannung über das Poti eingestellt wird.

Bei den relativ schnellen Geschwindigkeiten, die wir verwenden, muss die Versorgungsspannung unbedingt größer gewählt werden, weil eine Spule den Strom beim Einschalten nicht sofort annimmt. Bei einer höheren Spannung geht das schneller. Dabei muss man beachten, dass die Spannung ständig ein und ausgeschaltet wird, wenn sich der Motor dreht. Wenn die Spannung zu gering ist, dann gehen Schritte verloren.

⇒ **Die Schaltung sollte daher zwischen 14V (mindestens) und ca. 18V betrieben werden.**

Beispiele für am Modul einzustellende Spannungen:

Modul	NEMA 23 flach	NEMA 17	Spielzeug Schrittmotor	NEMA17 mit 17:1 Getriebe
TMC2208	0.628V	1.20V	0.080V	0.314V
TMC2100	0.628V	1.20V	0.080V	0.314V
DRV8825	0.320V	0.60V	0.035V	0.16V
A4988	0.512V	0.96V	64mV	0.128V

Vor dem Kalibriervorgang müssen die notwendigen Anpassungen der Konfiguration an die eigene Drehscheibe in der **Turntable_Config.h**-Datei vorgenommen werden. Änderungen in der Turntable.ino würden bei einem zukünftigen Software-Update verloren gehen.

Aus der Vielzahl der Konfigurationsvarialen hier die für mich, als Märklin-Nutzer, wesentlichen:

- Anzahl der benötigten Ports (in meinem Fall sind vier Ports vorgesehen) #define PORT_CNT
- Erste verwendete DCC Adresse eingeben, wenn man von dem im Programm vorgegebenen DCC_Adressraum abweichen will. #define FIRST_USED_DCC_ADDR
- Letzte verwendete DCC Adresse eingeben, wenn man nicht alle vorgesehenen DCC-Adressen benötigt und vom vorgegebenen DCC_Adressraum abweicht. #define LAST_USED_DCC_ADDR
- DCC_Port_Address_List anpassen #define DCC_PORT_ADDR_LIST
- Polarisation ein/ausschalten je nach System #define POLARISATION_RELAIS_PIN A1 *Polarisation Relais for dual rail system (Set to -1 if not used) * Zur Kalibrierung den DEBUG-Mode*

einschalten, damit im seriellen Monitor die Werte ausgelesen werden können #define ENABLE_DPRINTF 1 Debug Ausgaben ein
- Bei Bedarf Einstellungen der Ausrichtungen bzw Drehrichtung von Drehscheibe, Potentiometer, Dreh/Drückknopf und Display vornehmen
- Bei Bedarf Änderungen an den Einstellungen der verschiedenen Drehgeschwindigkeiten vornehmen.

Beispiel meiner Turntable_Config.h Datei.

Kalibriervorgang

Ich habe für einen ersten Test eine Scheibe mit vier Ports gewählt. Die Software berechnet dann automatisch vier symmetrische Ports. Wenn man einen Port exakt eingestellt hat und die Qualität der Scheibe gut ist, muss man die anderen Ports nicht mehr anpassen. Die Einstellung wird automatisch angepasst.

Zur Kalibrierung muss der (Test-)Aufbau abgeschlossen sein, d.h. Hall-Sensor/Magnet und die Gleise müssen positioniert sein. Die Stromversorgung für den Stepper und der NANO über den USB-Port sind angeschlossen. Der USB-Anschluss für den NANO wird nur für die Ausgabe über der seriellen Monitor während der Kalibrierung benötigt, nicht zum normalen Betrieb. Über die ARDUINO IDE den seriellen Monitor mit der Einstellung „Neue Zeile und 9600 Baud“ starten.

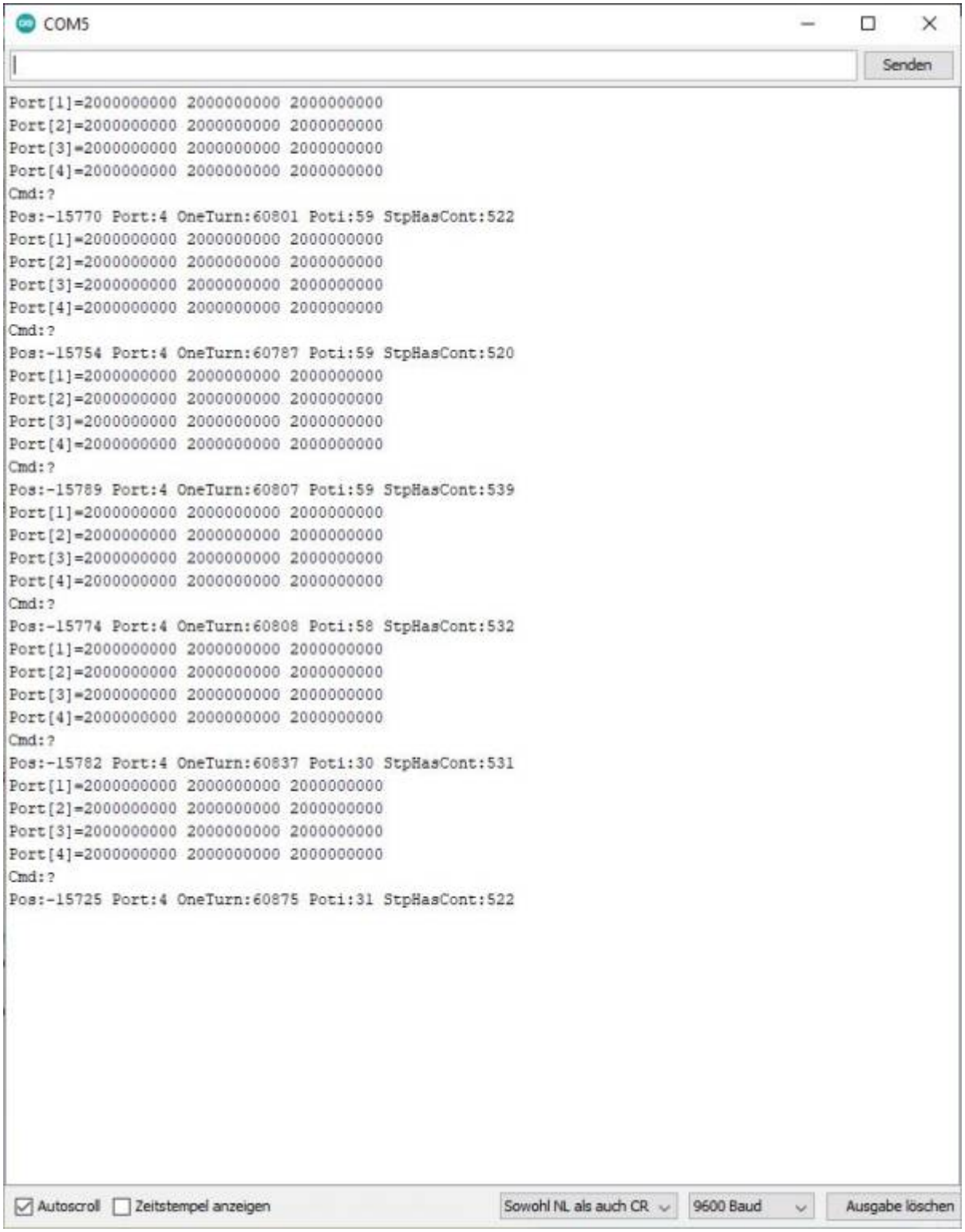
Da es wichtig ist, dass möglichst keine Microsteps verloren gehen, muss zunächst mehrfach geprüft werden wieviel Spiel die Drehscheibe hat.

- Über den Drehimpulsgeber im Menü „Reset all“ auswählen. Damit wird das EEPROM gelöscht und die Anzahl der benötigten Microsteps ermittelt.
- Nachdem der Vorgang abgeschlossen ist, über den seriellen Monitor ein „?“ senden.

```
         Angezeigt werden im seriellen Monitor:
                    * die Position an der sich die Drehscheibe befindet,
                    * die im Programm automatisch gesetzte Port-Nummer für
diese Position,
                    * die Anzahl der Microsteps für OneTurn,
                    * die Einstellung des Poti und das Spiel durch die
Anzahl der Microsteps (StpHasCont = Stepps has Contact)
```

- Über den Drehimpulsgeber im Menü erneut „Reset all“ auswählen.
- Diesen Vorgang mehrfach wiederholen, um verschiedene Ergebnisse zu erhalten.

Bei mir sieht das Ergebnis so aus:



Liefern „OneTurn“ und „StpHasCont“ immer das gleiche Ergebnis, herzlichen Glückwunsch! Der Antrieb der Drehscheibe ist von guter Qualität.

Bei meiner Konstruktion ist das erwartungsgemäß nicht der Fall. Ich habe ein Spiel von ca. 3.1 °. Aber auch dieses recht große Spiel kann das Programm automatisch berücksichtigen und korrigieren. Wenn man in der einen Richtung an einen Port fährt, sollte das Programm automatisch das Spiel berücksichtigen und die notwendigen Microsteps mehr ausführen als beim Anfahren aus der anderen Richtung. Dazu werden die Ports von beiden Seiten aus angefahren und die Positionen gespeichert.

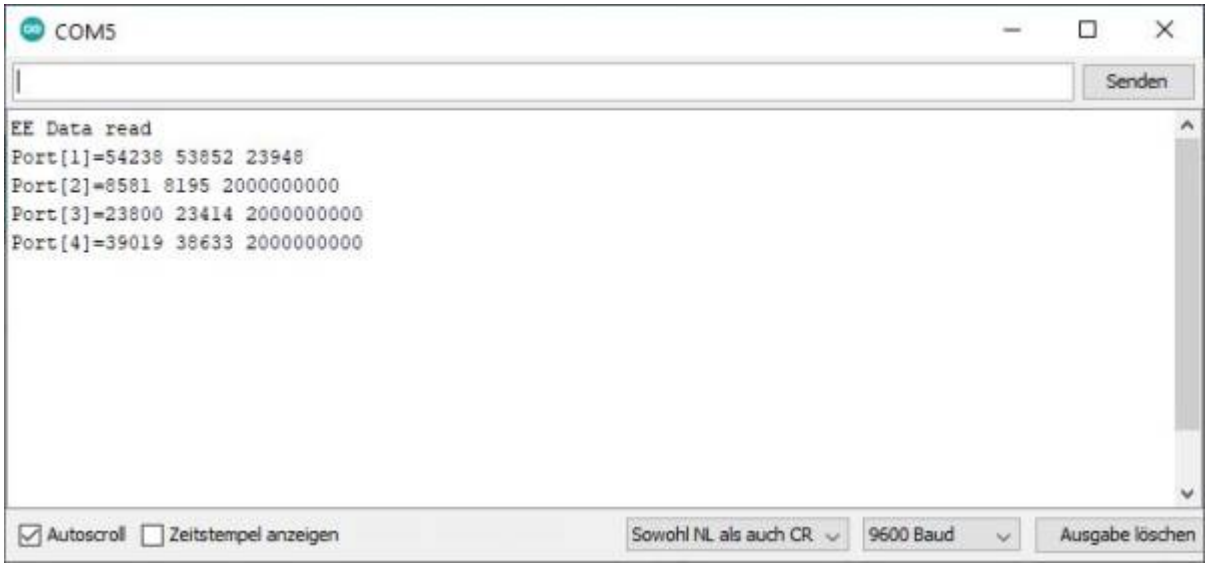
Wenn man das für Anschluss 1 machst, dann bekommen zunächst alle anderen Ports den gleichen Korrekturfaktor. In der „EE Data“ Tabelle ist dann die zweite Spalte ausgefüllt (Nicht 2000000000).

Die dritte Spalte zeigt den Wert für „reverse“ an, eine 180 Grad Drehung. Auch diesen Wert kann man separat speichern.

Hört sich kompliziert an, ist aber mit etwas Konzentration recht leicht durchzuführen:

1. Die Drehscheibe steht nach dem obigen mehrfachen Reset auf Port 4. Ich will Port 1 einstellen.
2. Mit dem Poti die Drehscheibe in Richtung Port 1 drehen. Dabei immer diese Drehrichtung beibehalten und auf keinen Fall zurückdrehen, da dann das Spiel beim Richtungswechsel ein exaktes Ergebnis verhindert und sich die Position nicht speichern lässt. Zum exaktem Ausrichten auf den letzten Millimetern kann auch über das Menü die Funktion „Move manual“ ausgewählt werden und die Drehscheibe in Microsteps bewegt werden. Aber auch hier nur in eine Richtung. Ist man über das Ziel hinausgeschossen, den ganzen Vorgang von Port 4 aus wiederholen.
3. Ist die Drehscheibe exakt positioniert über das Menü „Save Position“ auswählen.
4. Es erscheint „Select port to be saved“, nun über den Dreh/Drückknopf die gewünschte Port-Nummer auswählen, hier die „1“, und zur Bestätigung erscheint „Position saved to port 1“.
5. Nun zu Port 2 wechseln bzw. zu einer Position, die auf der anderen Seite von Port 1 liegt, um wie oben beschrieben, die Port-Position von beiden Seiten aus zu sichern.
6. Die Vorgänge wie unter 2 – 4 beschrieben auch von dieser Seite durchführen.
7. Nach dem Erreichen der exakten Port 1 Position diese wieder sichern. Im Menü erscheint dann „Update Ports?“ mit dem Untermenü „ Only this“ und „All port“. In diesem Fall „Only this“ auswählen. Mit der Funktion „All ports“ werden die gespeicherten Werte für alle Ports korrigiert.
8. Mit den anderen Port entsprechend verfahren.
9. Über den Menüpunkt „Reverse“ dreht sich die Drehscheibe um 180 Grad. Im Display wird „~3 und 1 reverse“ angezeigt. Die Tilde zeigt an, dass sich die Drehscheibe im Bereich von Port 3 befindet, jedoch nicht exakt positioniert ist.
10. Wieder den Menüpunkt „Save“ anklicken. Es stehen drei Möglichkeiten zur Auswahl: „Reverse side“, „Normal side“ und „Abort“. Nach dem Anklicken von „Reverse side“ kommt man wieder zur Auswahl „Only this“ und „All ports“. Nach dem Anklicken von „Only this“ erhält man die Meldung „Position saved to port 1“

Damit ist die Einstellung für Port 1 durchgeführt und die anderen Ports können bei Bedarf entsprechend kalibriert werden. Durch Drücken der Reset-Taste des NANO wird das EEPROM ausgelesen und die Werte für die Ports angezeigt.



Ergänzungen

Der Nullpunkt wird immer dann neu gesetzt, wenn die Scheibe in positiver Richtung am Hall-Sensor = Nullpunkt vorbeikommt. Die positive Richtung ist die Richtung welche beim Re-Kalibrieren zum Start verwendet wird.

Wenn sie in negativer Richtung am Hall-Sensor vorbeikommt, dann wird der Nullpunkt nur dann neu bestimmt, wenn sie bereits eine Umdrehung in negativer Richtung gedreht wurde.

Die Kalibrierung wird aber auch dann nur in Positiver Richtung vorgenommen. Darum dreht sie sich zunächst ein Stück zurück bevor die Kalibrierung in positiver Richtung beginnt. Dabei wird „Turn back and set zero pos.“ angezeigt.

Steuerung der Drehscheibe

Die Drehscheibe läßt z.Zt. über Poti/Dreh/Drückknopf und DCC-Befehle steuern. Ist der NANO mit der ARDUINO IDE verbunden, kann die Steuerung eingeschränkt auch über den seriellen Monitor erfolgen.

Steuerung über DCC

In der Turntabel.ino Datei sind folgende DCC-Adressen voreingestellt. Die Anpassung an die eigene Anlage sollte in der Turntable_config.h -Datei vorgenommen werden.

```
  // Use negativ addresses to disable the corrosponding command
#ifndef DCC_DISABLE_SOUND_ADDR
#define DCC_DISABLE_SOUND_ADDR              DCC_ADD_DIR(214, RED)       //
Disable the automatic generated sound if the turntable starts/stops moving
#endif
#ifndef DCC_ENABLE_SOUND_ADDR
#define DCC_ENABLE_SOUND_ADDR               DCC_ADD_DIR(214, GRN)       //
Enable the automatic generated sound if the turntable starts/stops moving
#endif
#ifndef DCC_VOLUME_DN_ADDR
#define DCC_VOLUME_DN_ADDR                  DCC_ADD_DIR(215, RED)       //
decrease the volume
#endif
#ifndef DCC_VOLUME_UP_ADDR
#define DCC_VOLUME_UP_ADDR                  DCC_ADD_DIR(215, GRN)       //
increase the volume
#endif
#ifndef DCC_VOLUME_1_ADDR
#define DCC_VOLUME_1_ADDR                   DCC_ADD_DIR(216, RED)       //
Set the sound volume to SOUND_VOLUME1 (10 by default)
#endif
#ifndef DCC_VOLUME_2_ADDR
#define DCC_VOLUME_2_ADDR                   DCC_ADD_DIR(216, GRN)       //
Set the sound volume to SOUND_VOLUME2 (20 by default)
```

```
#endif

#ifndef DCC_PLAY_SOUND1_ADDR
#define DCC_PLAY_SOUND1_ADDR                DCC_ADD_DIR(217, RED)       //
Play sound 1
#endif
#ifndef DCC_PLAY_SOUND2_ADDR
#define DCC_PLAY_SOUND2_ADDR                DCC_ADD_DIR(217, GRN)       //
Play sound 2
#endif
#ifndef DCC_PLAY_SOUND3_ADDR
#define DCC_PLAY_SOUND3_ADDR                DCC_ADD_DIR(218, RED)       //
Play sound 3
#endif
#ifndef DCC_PLAY_SOUND4_ADDR
#define DCC_PLAY_SOUND4_ADDR                DCC_ADD_DIR(218, GRN)       //
Play sound 4
#endif
#ifndef DCC_PLAY_SOUND5_ADDR
#define DCC_PLAY_SOUND5_ADDR                DCC_ADD_DIR(219, RED)       //
Play sound 5
#endif
#ifndef DCC_PLAY_SOUND6_ADDR
#define DCC_PLAY_SOUND6_ADDR                DCC_ADD_DIR(219, GRN)       //
Play sound 6
#endif
#ifndef DCC_PLAY_SOUND7_ADDR
#define DCC_PLAY_SOUND7_ADDR                DCC_ADD_DIR(220, RED)       //
Play sound 7
#endif
#ifndef DCC_PLAY_SOUND8_ADDR
#define DCC_PLAY_SOUND8_ADDR                DCC_ADD_DIR(220, GRN)       //
Play sound 8
#endif
#ifndef DCC_DISABLE_LIGHT_ADDR
#define DCC_DISABLE_LIGHT_ADDR              DCC_ADD_DIR(221, RED)       //
Disable the light in the machine house on the turntable
#endif
#ifndef DCC_ENABLE_LIGHT_ADDR
#define DCC_ENABLE_LIGHT_ADDR               DCC_ADD_DIR(221, GRN)       //
Enable the light
#endif
#ifndef DCC_SET_SPEED1_ADDR
#define DCC_SET_SPEED1_ADDR                 DCC_ADD_DIR(222, RED)       //
Set the moving speed to MOVE_SPEED1
#endif
#ifndef DCC_SET_SPEED2_ADDR
#define DCC_SET_SPEED2_ADDR                 DCC_ADD_DIR(222, GRN)       //
Set the moving speed to MOVE_SPEED2
#endif
```

```
#ifndef DCC_SET_SPEED3_ADDR
#define DCC_SET_SPEED3_ADDR                DCC_ADD_DIR(223, RED)       //
Set the moving speed to MOVE_SPEED3
#endif
#ifndef DCC_SET_SPEED4_ADDR
#define DCC_SET_SPEED4_ADDR                DCC_ADD_DIR(223, GRN)       //
Set the moving speed to MOVE_SPEED4
#endif
#ifndef DCC_STEP_POS_DIR_ADDR
#define DCC_STEP_POS_DIR_ADDR              DCC_ADD_DIR(224, RED)       //
Turn to the next port in positive direction
#endif
#ifndef DCC_STEP_NEG_DIR_ADDR
#define DCC_STEP_NEG_DIR_ADDR              DCC_ADD_DIR(224, GRN)       //
Turn to the next port in negative direction
#endif
#ifndef DCC_ROTATE_POS_DIR_ADDR
#define DCC_ROTATE_POS_DIR_ADDR            DCC_ADD_DIR(225, RED)       //
Continiously rotate in the positive direction
#endif
#ifndef DCC_ROTATE_NEG_DIR_ADDR
#define DCC_ROTATE_NEG_DIR_ADDR            DCC_ADD_DIR(225, GRN)       //
Continiously rotate in the negative direction
#endif
#ifndef DCC_STOPP_ADDR
#define DCC_STOPP_ADDR                     DCC_ADD_DIR(226, RED)       //
Stop the turntable
#endif
#ifndef DCC_CALIBRATE_ADDR
#define DCC_CALIBRATE_ADDR                 DCC_ADD_DIR(226, GRN)       //
Calibrate the zero position (During the calibration no other commands are
accepted)
#endif
#ifndef DCC_SET_POL_REL0_ADDR
#define DCC_SET_POL_REL0_ADDR              DCC_ADD_DIR(227, RED)       //
Set the polarisation to RED and disable the automatic
#endif
#ifndef DCC_SET_POL_REL1_ADDR
#define DCC_SET_POL_REL1_ADDR              DCC_ADD_DIR(227, GRN)       //
Set the polarisation to GRN and disable the automatic
#endif
#ifndef DCC_AUTO_POL_REL_ADDR
#define DCC_AUTO_POL_REL_ADDR              DCC_ADD_DIR(228, RED)       //
Enable the automatic polarisation mode. The releai wil be set after the next
move
#endif
#ifndef DCC_REVERSE_TABLE_ADDR
#define DCC_REVERSE_TABLE_ADDR             DCC_ADD_DIR(228, GRN)       //
Reverse the turntabel
#endif
```

```
// List of DCC adresses to move to the desired port. Must contain exact
PORT_CNT entries. If not a compiler error is generated.
#ifndef DCC_PORT_ADDR_LIST
#define DCC_PORT_ADDR_LIST                  DCC_PORT_ADDR(1, 229, RED), \
                                            DCC_PORT_ADDR(2, 229, GRN), \
                                            DCC_PORT_ADDR(3, 230, RED), \
                                            DCC_PORT_ADDR(4, 230, GRN), \
                                            DCC_PORT_ADDR(5, 231, RED), \
                                            DCC_PORT_ADDR(6, 231, GRN), \
                                            DCC_PORT_ADDR(7, 232, RED), \
                                            DCC_PORT_ADDR(8, 232, GRN), \
                                            DCC_PORT_ADDR(9, 233, RED), \
                                            DCC_PORT_ADDR(10, 233, GRN), \
                                            DCC_PORT_ADDR(11, 234, RED), \
                                            DCC_PORT_ADDR(12, 234, GRN), \
                                            DCC_PORT_ADDR(13, 235, RED), \
                                            DCC_PORT_ADDR(14, 235, GRN), \
                                            DCC_PORT_ADDR(15, 236, RED), \
                                            DCC_PORT_ADDR(16, 236, GRN), \
                                            DCC_PORT_ADDR(17, 237, RED), \
                                            DCC_PORT_ADDR(18, 237, GRN), \
                                            DCC_PORT_ADDR(19, 238, RED), \
                                            DCC_PORT_ADDR(20, 238, GRN), \
                                            DCC_PORT_ADDR(21, 239, RED), \
                                            DCC_PORT_ADDR(22, 239, GRN), \
                                            DCC_PORT_ADDR(23, 240, RED), \
                                            DCC_PORT_ADDR(24, 240, GRN),
#endif

#ifndef LAST_USED_DCC_ADDR                                                  //
Must be set to the last used address
#define LAST_USED_DCC_ADDR                  DCC_CHKADDR(230, GRN)       //
If wrong limmits are used an "warning: division by zero" will be generated
#endif
```

Steuerung über den seriellen Monitor

```
   m-1000  Move 1000 micro steps counter clock wise
   m+1000  Move 1000 micro steps clock wise
   p+123   Move to step position +123
   s5000   Set the speed to 5000
   ?       Print position
   w+2     Write port position 2 for the positive turning direction (w-2
write the neg. direction)
   ce      Clear EEProm and restart
   +       Next Port
   -       Prior Port
   7       Move to Port 7
   r       Reverse turn Table
```

```
   o       Sound On/Off
```

Meine Turntable_Config.h Datei

```
// Configuration for the stepper program
//
// Add all individual config lines in this file
//   Example:
//   #define PORT_CNT                        24          // Number of ports
(Maximal 80)

#define ALLWAYS_CHECK_STEPS_ONE_TURN        0           // Always check the
steps for one turn at power on
#define PORT_CNT                            4           // Number of ports
#define CIRCUMFERENCE                       527.84      // 178 mm * Pi =
circumference of the turntable [mm]
#define ROTATIONSWITCH_DIRECTION            -1           // Set from 1 to -1
to change the direction of the rotation switch
#define ROTATIONSWITCH_MENU_DIR             -1          // Set from 1 to -1
to change the direction of the rotation switch in the menu
#define TURNTABLE_DIRECTION                 1           // Set to -1 to
change the rotation / port number direction
#define TURNBACK_SPEED                      15000       // Speed used for
TurnBackAndSetZero
#define NOT_ENABLE_PIN                      -1           // Set to -1 if the
stepper driver has an automatic power mode like the TMC2100
                                                        // The pin of the
module must be left open (std 6)
#define STEPPER_RAMP_LENGTH                 100         // Steps to speed up
the stepper to prevent loosing steps
                                                            // Set to 50 if
1/16 steps are used (MS1 - MS3 connected do +5V)
#define MOVE_SPEED1                         5000       // Default speed and
activated when DCC_SET_SPEED1_ADDR is received
#define MOVE_SPEED2                         4000       // Speed activated
when DCC_SET_SPEED2_ADDR is received
#define MOVE_SPEED3                         3000       // Speed activated
when DCC_SET_SPEED3_ADDR is received
#define MOVE_SPEED4                         2000        // Speed activated
when DCC_SET_SPEED4_ADDR is received
#define POLARISATION_RELAIS_PIN             -1          // Polarisation
Relais for dual rail system (Set to -1 if not used)
#define OLED_TYP                            91          // Tested with the
following displays
#define MOVING_FLASH_INVERS                 1           // Normal: 0 = LED
connected to GND
#define MOVING_FLASH_MODE                   2           // 1 = Blink, 2
double flash
#define ENABLE_DPRINTF              1        //Debug Ausgaben ein
#define SPEED_POTI_DIRECTION                1           // Set to -1 to
```

```
change the direction of the speed poti
#define SPEED_POTI_MID_RANGE                50         // Range of the speed
poti which is 0  (Old 50)
#define SPEED_POTI_CENTER                   512         // Center position
of the speed poti (Normaly 512)
#define ANALOG_SPEED_DIVISOR                30      // Divisor used to
calculate the analog speed with the poti (std 8, 50)
#define CLEARENCE_TEST_SPEED                25000       // Speed used in the
clearance test
#define CALIBRATE_SPEED                     25000       // Speed used for
the zero point and total number of stepps detection

#define STEP_PIN                            9        // New Pin 9, Testboard
4
#define FIRST_USED_DCC_ADDR                 DCC_CHKADDR(214, RED)        //
Used to speed up the program. Must be set to the first used address
#define DCC_PORT_ADDR_LIST                  DCC_PORT_ADDR(1, 229, RED), \
                        DCC_PORT_ADDR(2, 229, GRN), \
                                            DCC_PORT_ADDR(3, 230, RED), \
                                            DCC_PORT_ADDR(4, 230, GRN)

#define LAST_USED_DCC_ADDR                  DCC_CHKADDR(230, GRN)       //
If wrong limmits are used an "warning: division by zero" will be generated
```