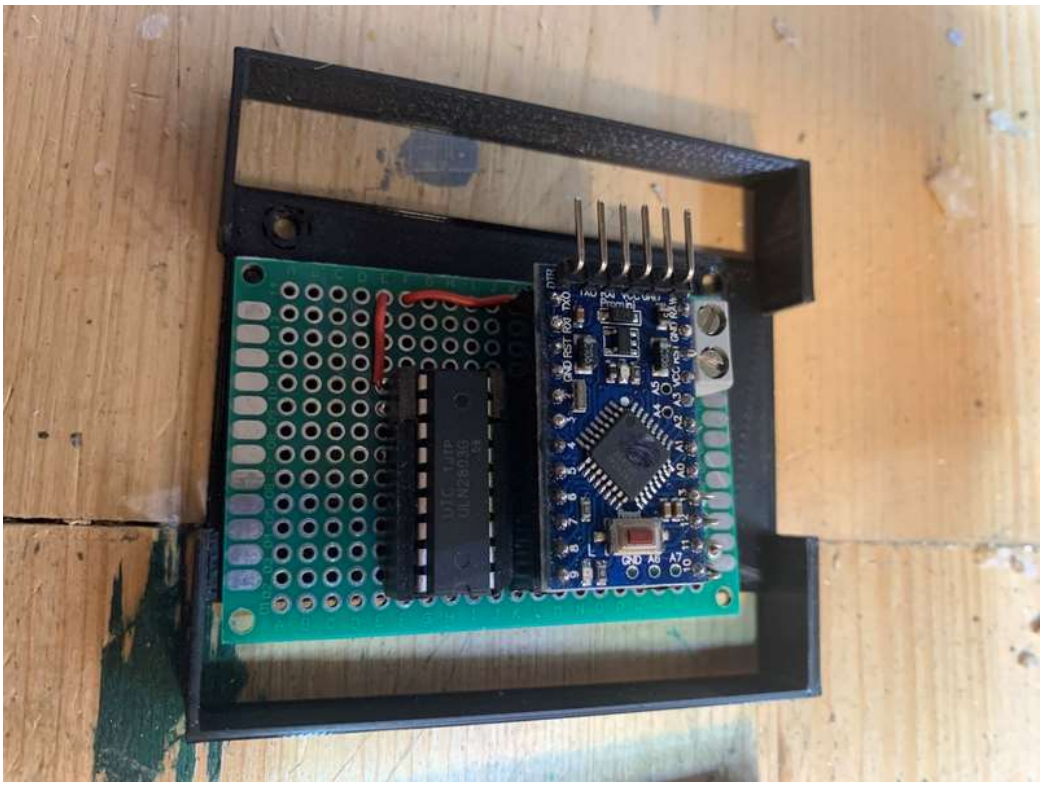
Arduino: Licht und Action

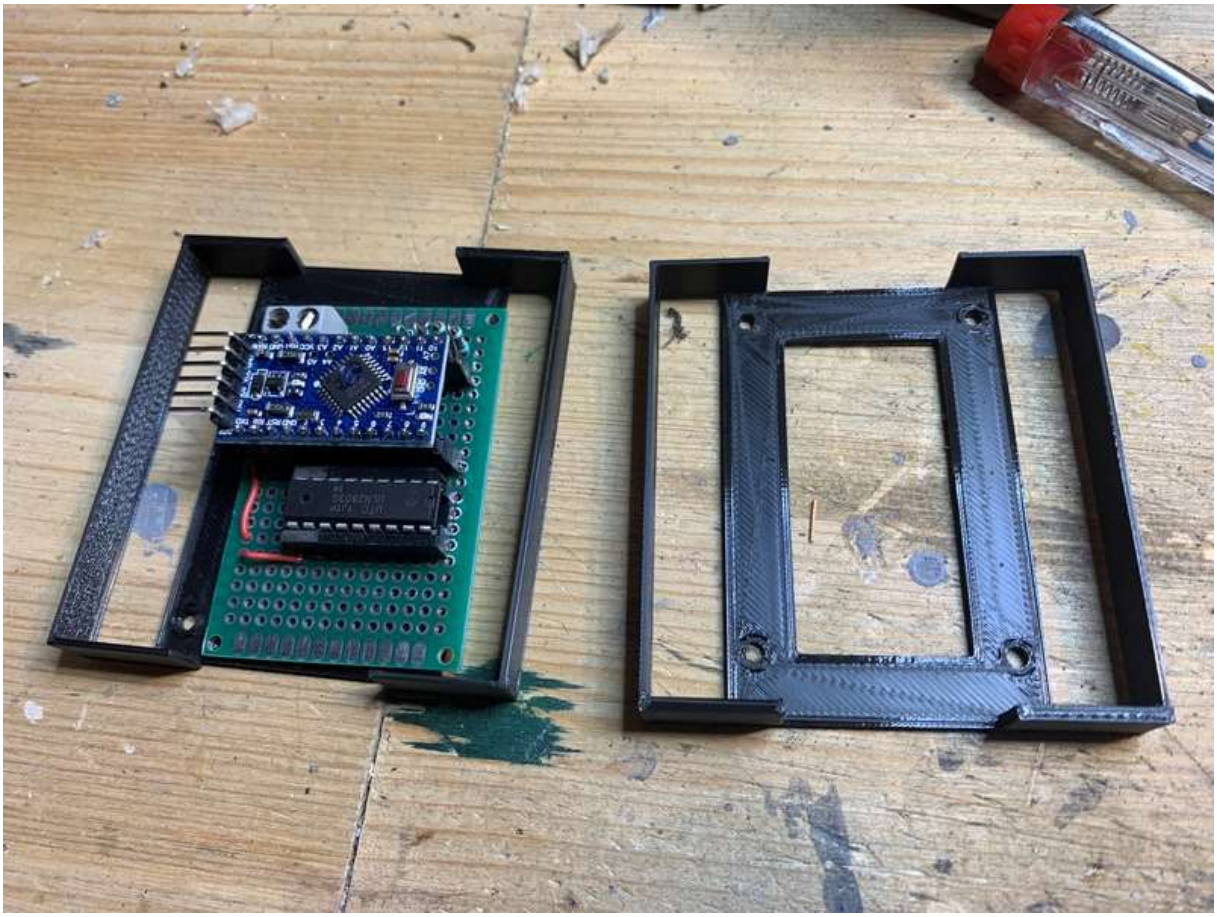
Dr.-Ing. Wolfgang Kreinberg

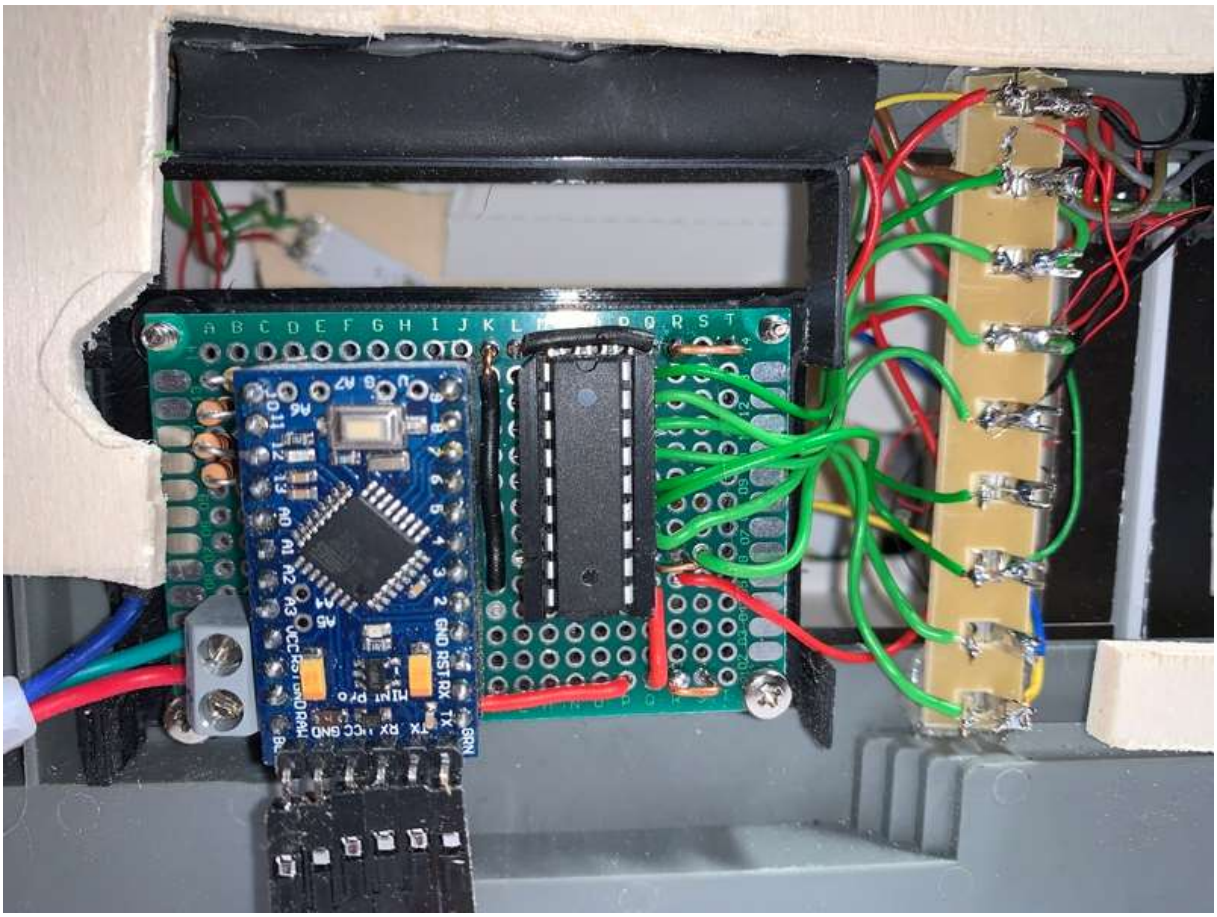
Seit Jahren nutze ich Arduino Micro Controller, seit Weihnachten 2019 einen 3D Drucker.
In Gebäude, Schuppen, Bahnhöfe setze ich Nanos oder Micro Pro.
Letztere haben keinen USB Port, aber sonst alle Möglichkeiten des größeren UNO.
Ich habe mir einen Standard gebastelt.

- Versorgung über 12 V DC
- 8-fach Darlington Array an Digital Ausgänge 2 bis 9 (kurze Wege)
- Arduino und Darlington auf Fassungen.
- Licht mit 12 V DC verdrahteten LED mit Viessmann Boxen bzw. selbst gedruckten Lichtkästen
- Licht an „abgeschrittene“ LED Strips 12 V
- 1 K Widerstände an Digitalausgänge 10 bis 13 (Einzel LED gegen Masse schalten)
- alles auf kleiner Platine
- selbst gedruckter kleiner Rahmen, passt in Bodenplatte von Faller Häusern

Alles wird mit 12 V DC versorgt und per Steckverbindung angeschlossen. Dadurch sind Test am PC und Wechsel auf die Anlage leichter.







Damit habe ich z.B. Lokschuppen ausgestattet (Tore mit Servo, Licht, blaue LED für Schweissflackern , ...)

Es geht weiter mit folgender Aufgabe

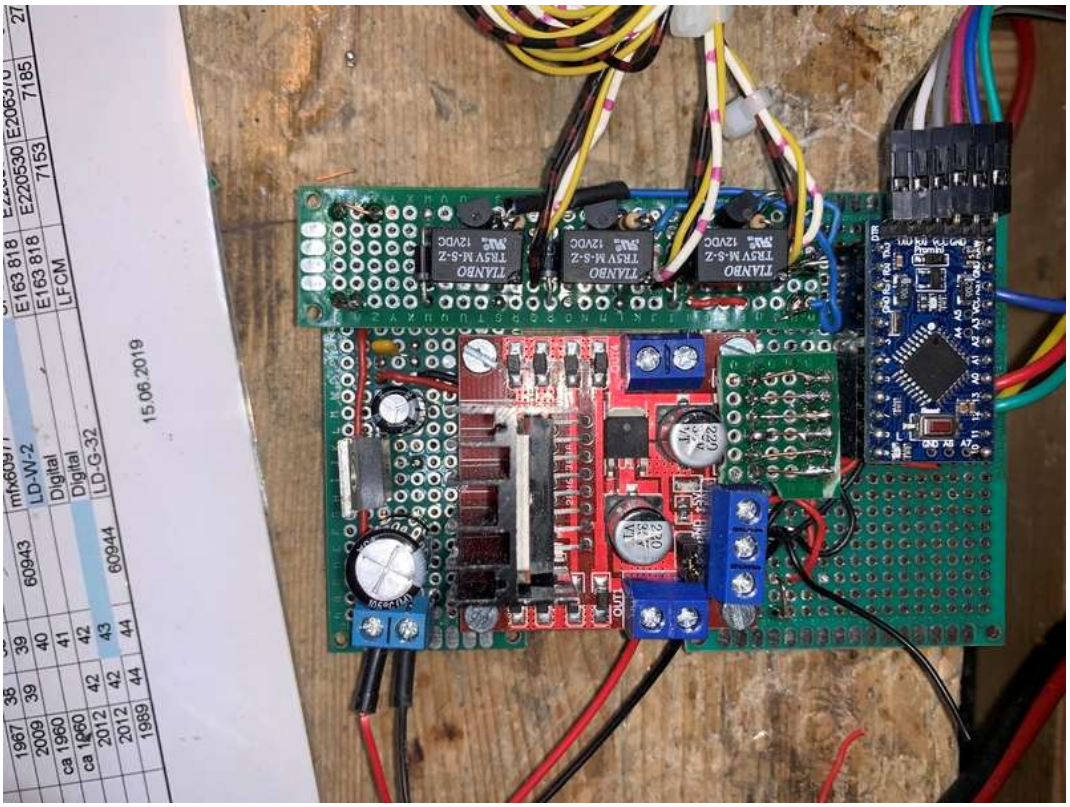
- Getriebemotor in Fallerkran
- Führerhausbeleuchtung
- Blinklicht bei Drehbewegung
- Krankabinenbeleuchtung
- Alles auf Knopfdruck und Zufallsgesteuert

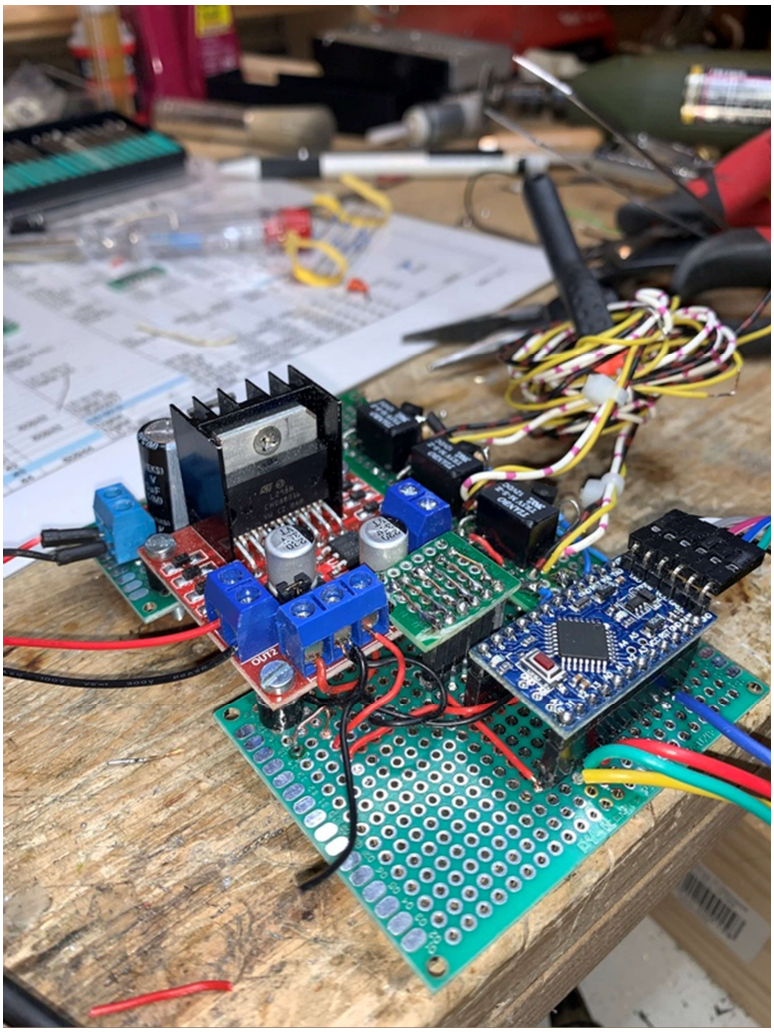
Dazu weiterhin

- alter Märklinkran auf Knopfdruck heben und drehen
- sowie Drehbewegung zufallsgesteuert

Wie soll das geschehen

- Arduino Pro Mini
- H-Bridge für zwei Motoren
- Drei 12 V Relais
- Drei LED
- Versorgungsspannung 12 V DC
- Runtergebrochen mit 7808 auf 8 V für H Bridge
- Aus der H-Bridge 5 V für Arduino
- 18 V AC für Märklin Kran über die Relais angesteuert
- Taster an A0 bis A3 für Aktionen.

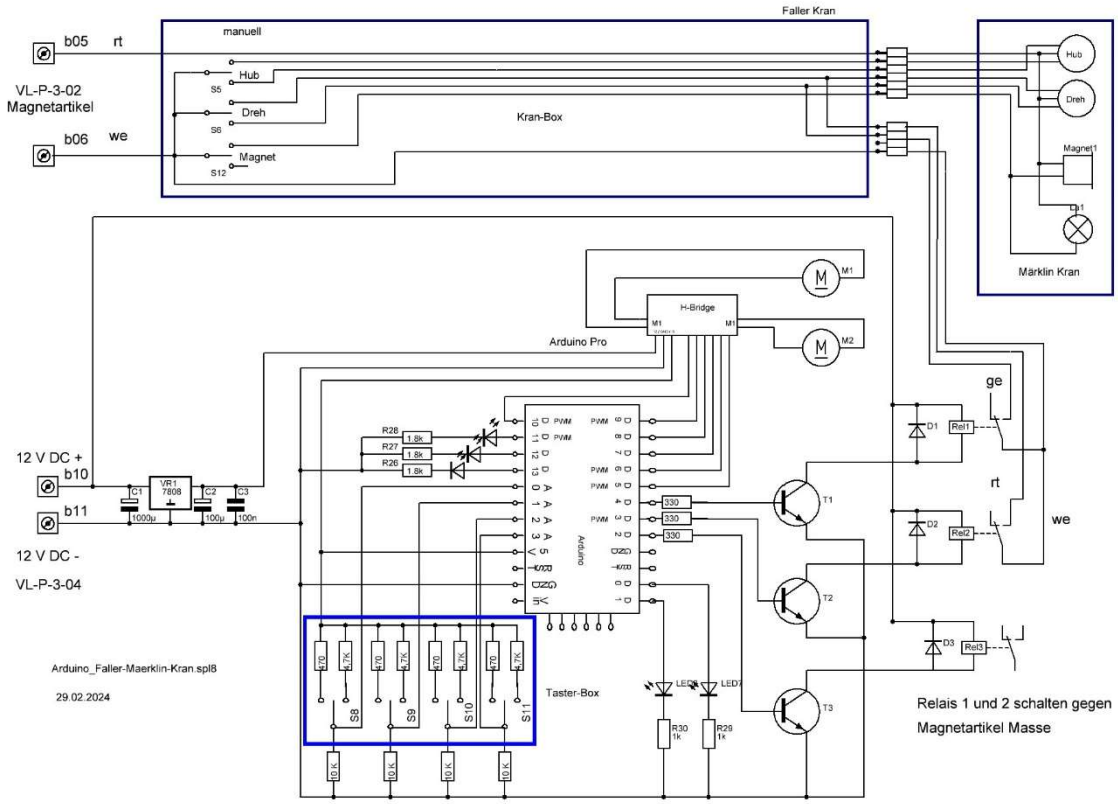




Programmierung des Arduino über PC und USB / TTL Adapter
Durch diesen Mischmasch an Aufgaben und Spannungen habe meine diskrete aber irgendwie standardisierte Lösung gewählt.

Zwei Digitalausgänge sind noch frei.

Die Halterung des Getriebemotors habe ich mit dem 3D Drucker gemacht.
Der Motor kommt von Conrad z.B. Bestell-Nr.: 1762749 - VQ



Hier der Schaltplan hinter dem Gewusel.

An den vier Analogeingängen können über einen rückstellenden Schalter neben dem Ruhezustand zwei weitere Zustände über die Eingangsspannung abgefragt werden.
Digitaleingänge:
0 und 1 steuern über Widerstände LEDs an
2 – 4 steuern über Transistoren die Relais an
5 - 7 sind für Motor 1
8 - 10 sind für Motor 2
11 - 13 steuern über Widerstände LEDs an

Versorgung:
12 V DC
- für die Relais
- über einen 7808 für die H-Bridge

5 V DC
- von der H-Bridge auf den Arduino

Schalter Märklin Kran

- manuell einschaltbar für Drehen, Heben und Magnet/Licht
- via Arduino für zufälliges Drehen

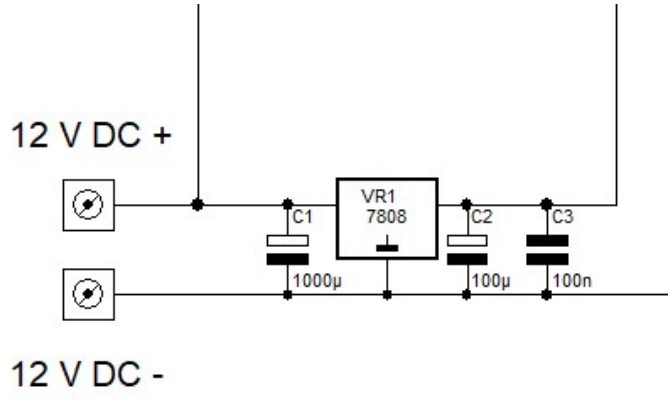
Da in der Schaltung einige Schaltprinzipien verwendet wurden, möchte ich sie jetzt Stück für Stück im Detail aufgreifen und erläutern.

Ich beginne mit der **Spannungsversorgung**.
Es werden folgende Spannungen benötigt:

12 V DC: für die Relais
8 V DC: für die H-Brücke zur Ansteuerung der kleinen Getriebe Motoren
5 V DC: für den Arduino
18 V AC: für anzusteuernde Modellbahnelemente, hier der "alte" Märklin Kran.

Die 12 V DC werden bei mir für jedes Anlagensegment durch Schaltnetzteile bereitgestellt, darauf gehe ich jetzt nicht weiter ein.

Beginnen wir mit den 8 V DC, das Detail ist dem obigen kompletten Schaltplan entnommen.



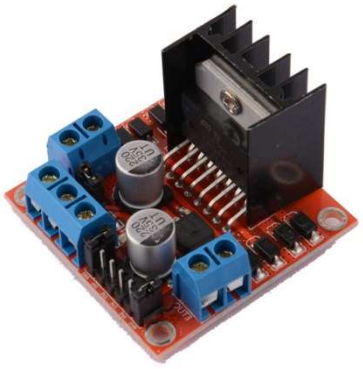
Die 12 V DC von dem Schaltnetzteil werden über eine Klemme eingespeist und über einen 7808 Spannungsregler auf 8 V DC gesenkt. Die beiden Elkos dienen zu Glättung, der 100 nF zur Siebung eventueller höherer Frequenzen.

Die 8 V DC werden dann der H Brücke zugeleitet. Dort geht es im nächsten Punkt weiter.

Nun geht es mit der **H-Brücke** weiter.
Die von mir verwendete H-Bridge hat zwei nette Features

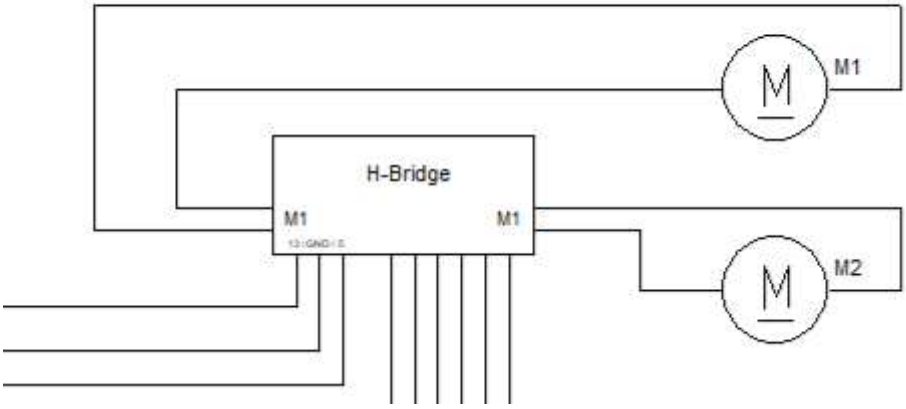
- Ansteuerung von zwei DC Motoren
- Spannungsstabilisator 5 V DC

Ich habe diese Brücken über Amazon bezogen:
Youmile 5er-Pack Dual H Bridge DC Stepper Motor Drive Controller Board Modul L298N für Arduino.



Links und rechts sind zwei Klemmen, hier kommen die 6-9 V Getriebemotoren dran.
Unten sind drei Klemmen
1: 18 V DC plus Eingang
2: 8V DC GND und 5 V DC GND
3: 5 V DC plus Ausgang

Im Schaltplan oben sieht das dann so aus:



Neben den Spannungsklemmen sieht man 3 x 2 Stifte, an die der Arduino angeschlossen wird.

Es geht weiter mit dem Anschluss an den **Arduino.**

Als Arduino läuft bei mir ein Arduino Mini Pro, die habe ich bei AZ Delivery bezogen:

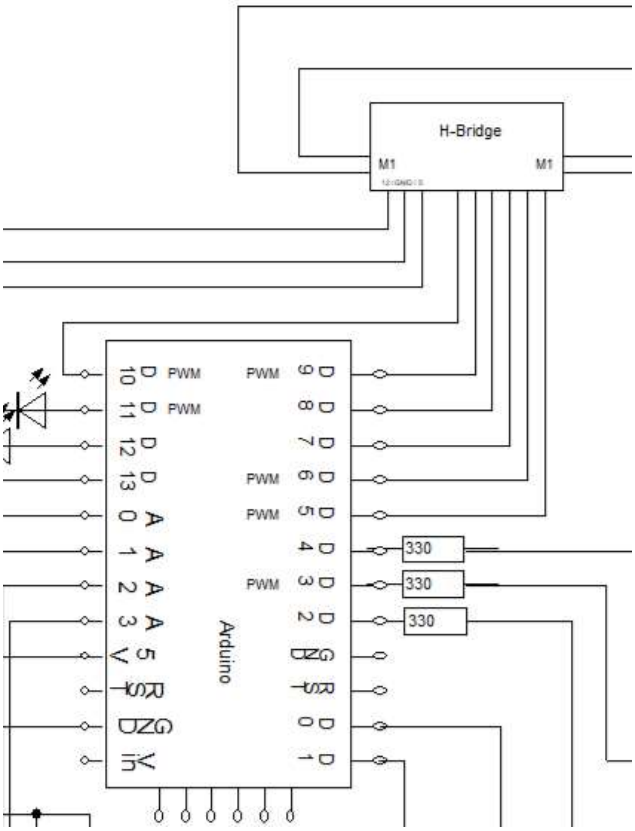
AZDelivery 3 x PRO MINI mit 5V ATmega328 und 16MHz

Es handelt sich um komplette Arduinos (wie ein UNO), aber sie sind kleiner und haben keinen direkten USB Anschluss. Wie sie trotzdem programmiert werden können, zeige ich dann noch.

Hier ist das Schaltungsdetail

Motor 1 geht an pins 10, 9 und 8
Motor 2 geht an pins 7, 6 und 5

Das Wichtige ist, dass von den 6 Stiften der H-Brücke der linke (Motor 1) und der rechte Pin (Motor 2) an einen PWM Eingang kommen, damit die Geschwindigkeit des Arduino geregelt werden kann. Pin 10 und 5 sind solche PWM taugliche Ausgänge.



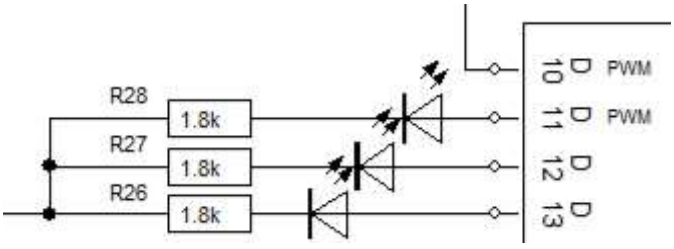
Weiterhin liegen alle schön nebeneinander, ich habe auf ein Stückchen Platine zwei Buchsenleisten (6 Kontakte) gelötet und auf Arduino und H-Bridge als Verbinder gesteckt. Das kann man ganz gut auf dem Gesamtbild weiter oben sehen.

Weiterhin geht eine Verbindung von den 5 V Plus und 5 V Ground der H-Bridge auf die 5 V und GND Pins des Arduino.

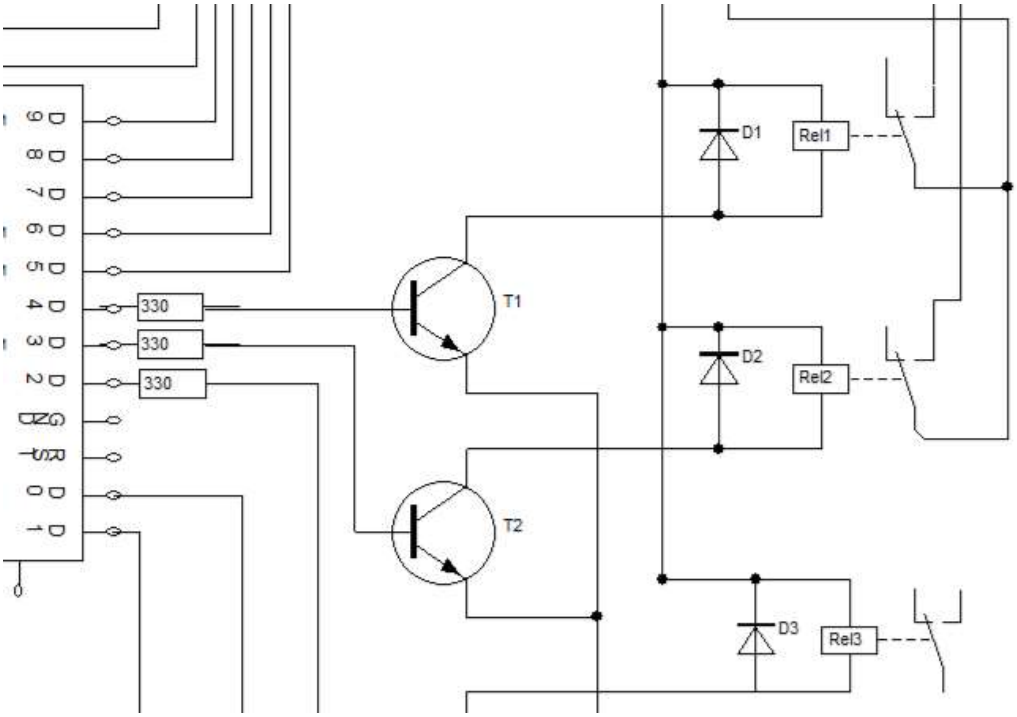
Weiter geht es mit den Ausgangsschaltungen zusätzlich zu der H-Bridge.

Hier nutze ich zwei:

1) Digitalausgang zur Ansteuerung einer LED
2) Digitalausgang zur Ansteuerung von 12 V Relais

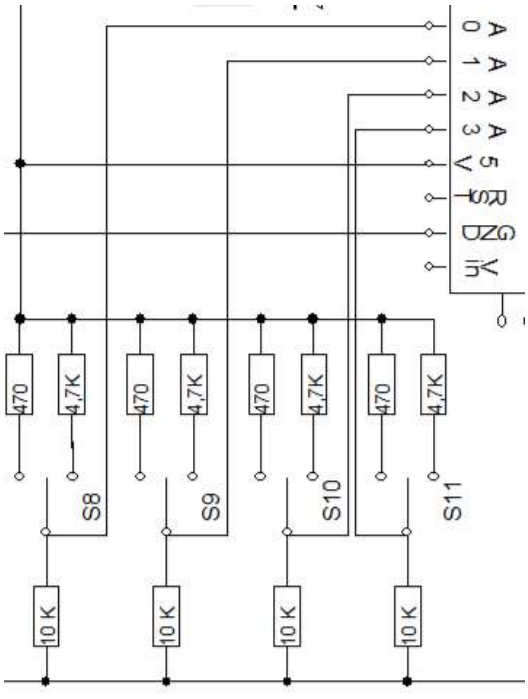


Ich habe drei LEDs an Pins 11, 12 und 13 gehängt.
Sie werden gegen GND geschaltet, leuchten also, wenn die Ausgänge auf HIGH gelegt werden.



Die Relais können nicht direkt an den Arduino gelegt werden, daher ist ein klassischer NPN Transistor dazwischen. Der Emitter liegt jeweils an GND.
Die Relais ziehen an, wenn der Ausgang des Arduino auf HIGH gelegt wird. Der Transistor schaltet durch und verbindet die Relais mit 12 V DC plus. Als Relais habe ich 1 mal UM gewählt, wer mehr Kontakte benötigt, hat hier die Wahl.
Über der Relaisspule liegt eine Freilaufdiode.

Weiter geht es mit den Analogen Eingängen.



Diese Eingänge benutze ich, um Taster abzufragen.
Die einfachste Methode ist es, einen Taster an einen Eingang zu legen. Das geht sowohl bei den analogen als auch bei den digitalen Pins.
Ich wollte aber in meinem Fall 8 "Taster" an 4 Pins legen.
Dazu nutze ich die Möglichkeiten der analogen Pins aus, eine Eingangsspannung abzufragen.

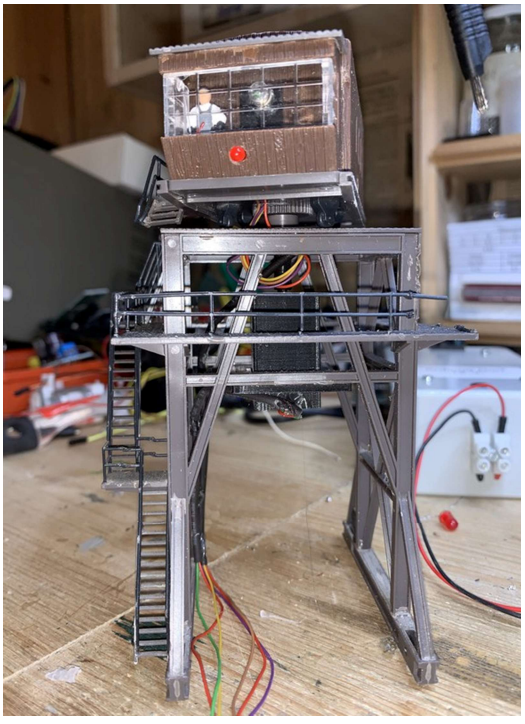
Die analogen Pins A0 bis A3 werden über den internen Pullup Widerstand "hochgezogen".
In dem Bild habe ich 4 Schalter mit selbststellender Mittelstellung eingezeichnet.
Bei Betätigung gehen die A0 bis A3 entweder über 470 Ohm oder über 4,7 KOhm gegen 5 V DC plus.

Gegen Masse liegt jeweils ein Widerstand von 10 kOhm.

Über die jeweiligen Spannungsteiler ergeben sich unterschiedliche Eingangsspannungen, die per Software abgefragt werden können
Nicht gedrückt: ca 2,5 V
470 Ohm gedrückt: über 3,5 V
4,7 K Ohm gedrückt: unter 2,5 V
Die genauen Werte zeige ich bei Erläuterung der Programmschritte. Der Vorteil dieser Schalter ist, dass technisch immer nur ein Widerstand geschaltet wird. Man kann natürlich auch zwei unabhängige Taster nehmen (wie ich noch zeige), dann können jedoch beide Taster gleichzeitig gedrückt werden und liefern eventuell unbestimmte Ergebnisse.

Nach diesen technischen Details hier noch einmal das Thema:

Links ist der "Faller Kran", es war ein Bausatz ohne Antrieb. Rechts ist der alte "Märklin Kran", der über Taster und Schalter klassisch betätigt wird. Das alte Schaltpult war verschwunden, ich musste einen Ersatz bauen. In der Mitte ist ein Wasserkran, da soll der zweite Motor oder en Servo später rein.



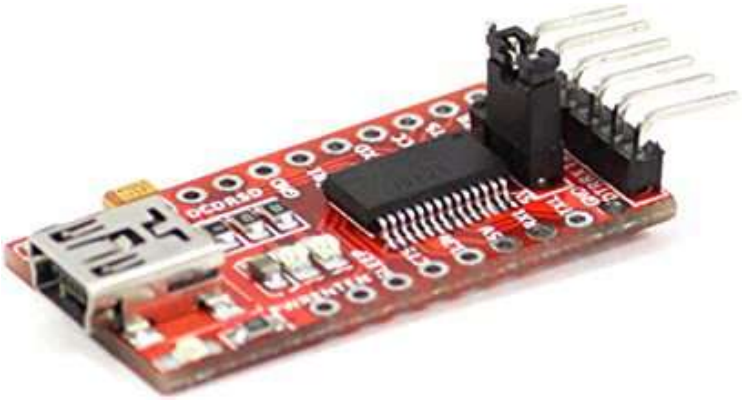
Hier steht der Faller Kran auf der Werkbank.

Ich habe folgende Erweiterungen gemacht
- Kranführer in Kanzel gesetzt
- weißes Licht in Führerkanzel
- rotes Licht (als Blinklicht) an Vorderseite
- weißes Licht in den Hauptraum des Krans
- Adapter in Kranhaus (gedruckt und durch das Loch nach unten geführt)
- Getriebemotor von unten angeschraubt (Achse steckt in Adapter)
- Gehäuse um den Motor herum gedruckt
- Zuleitungen (insgesamt 6) durch schwarze Schrumpfschläuche nach unten geführt

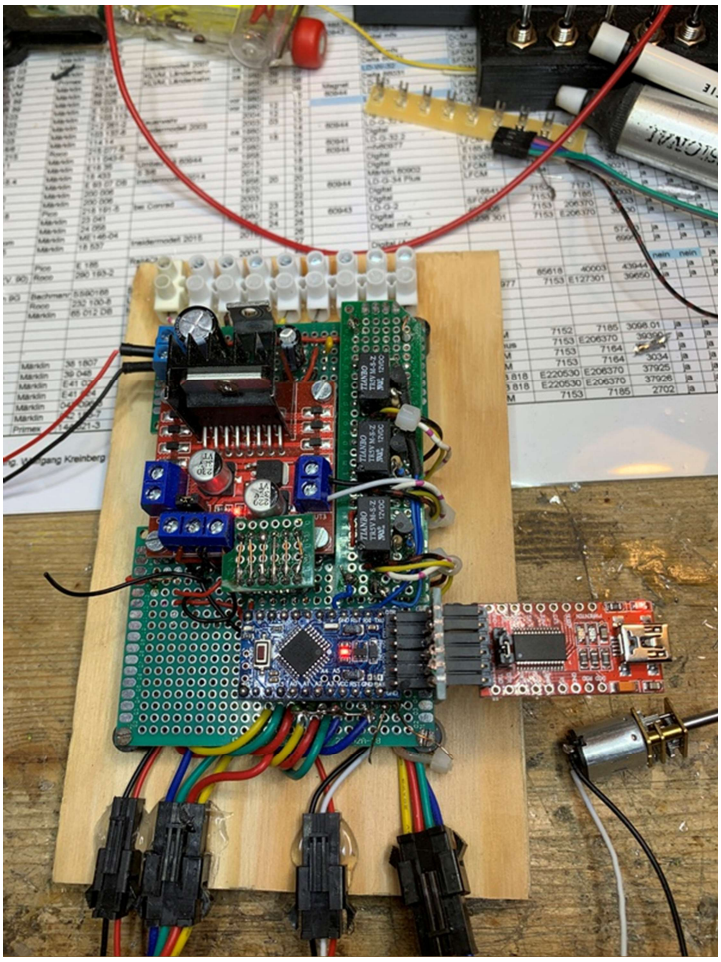
Es geht weiter mit dem Programmieren.

Ich nutze die Arduino IDE auf einem PC. Die Hauptarbeit passiert am Desktop, das Feintuning mit dem Laptop im Keller. Daher sind alle Arduinodaten auf einem USB Stick.

Da der Mini Pro keinen USB Anschluss hat, muss ein Adapter her. Ich nutze den FTDI Adapter FT232RL von AZ Delivery.

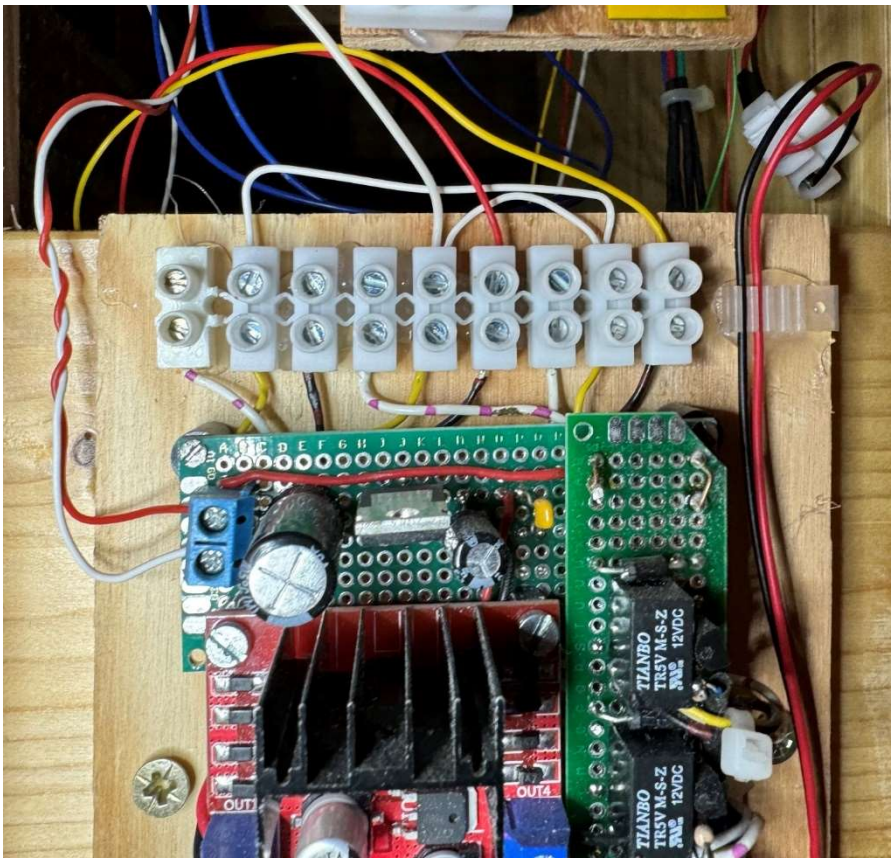


Bei den ersten Pro Mini habe ich den Fehler gemacht, alle mitgelieferten Pinleisten anzulöten. Für den Anschluss des FTDI Adapter prallen dann Stifte auf Stifte, also musste ich zunächst Kabel (Buchsen auf Buchsen) kaufen und dazwischen stecken.
Nun habe ich mir für die alten (Stifte auf Stifte) einen ähnlichen Adapter gelötet wie für die H-Bridge.



Man erkennt hier auch die vier- bzw. drei- bzw. zwei-poligen Steckverbindungen. Damit kann ich die angeschlossenen Elemente ^^abstöpseln und die Schaltung (auf eine Platte geschraubt) von der Anlage nehmen.

Bei meiner Pro Mini Lieferung hatte ich eine kleine Herausforderung zu lösen.
Obwohl es sich um 5 V Typen handelte, musste ich bei einigen den Jumper auf dem FTDI Adapter von 5 V auf 3,3 V umsetzen.



Oberhalb der Leiterplatte sind Klemmen 3 mal 3 für die Relaisausgänge

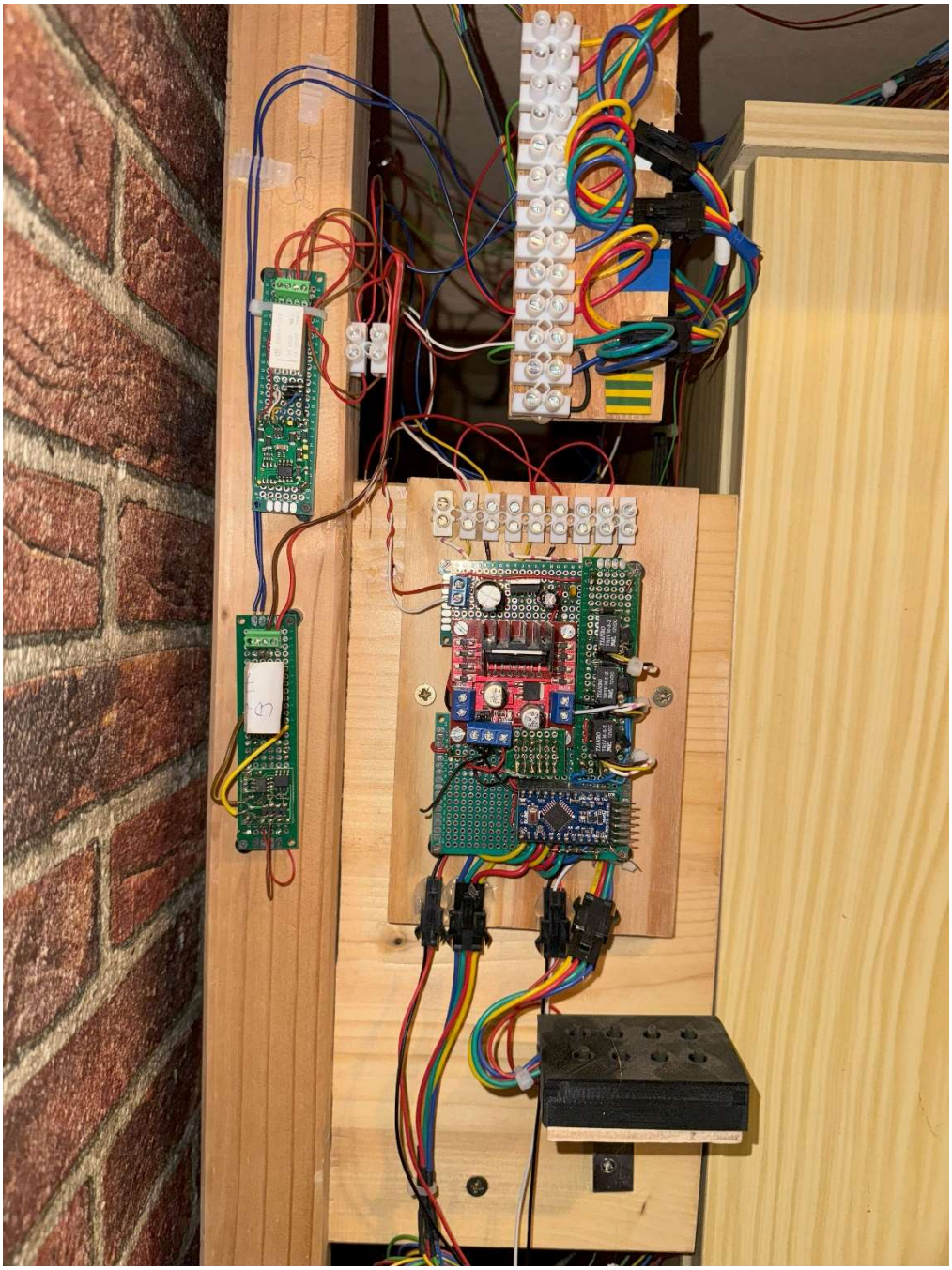
AK Arbeitskontakt
RK Ruhekontakt

Rel_3			Rel_2			Rel_1		
			Märklin Kran links			Märklin Kran rechts		
	we			we			we	
				Weichen Strom			Weichen Strom	
				we				
RK		AK	RK		AK	RK		AK
we/rt	ge	sw/li	we/rt	ge	sw/li	we/rt	ge	sw/li

Die Belegung in der Tasterbox

Faller links	Motor2 links	Märklin links	
A0	A1	A2	A3
Faller rechts	Motor2 rechts	Märklin rechts	Rel_3

Unter der Anlage ist die komplette Anordnung zu sehen.



Die Steckverbindungen sind von links nach rechts

- Faller Kran Motor
- Märklin Kran
- Motor 2
- Taster Box

Die obere Platine am senkrechten Pfosten ist ein Fernschalter auf der Basis eines Lokdecoders. Das bistabile Relais schaltet die 12 V DC Verbindung ein.

Auf in **Programmdetails**. In diesem Projekt ist bewusst nicht die MOBATOOLS.LIB enthalten, um verschiedene Techniken anwenden (und zeigen zu können)

Ich fang mal mit den beiden Schaltern / Tastern am Analog Port an.

Den kompletten Programmcode bringe ich dann noch in der korrekten Form, hier extrahiere ich erst einmal die relevanten Teile.

Zunächst die Definitionen:

```
// Analogeingänge
// An jedem Analogeingang sind zwei Taster
// Pull-up Widerstand eingeschaltet
// 10 K Widerstand gegen GND
// Taster a über 470 Ohm gegen Vcc
// Taster b über 4,7 K gegen Vcc
//
const int Taste_1 = A0; // Steuertaste 1 auf Analog 0 liefert integer 14
const int Taste_2 = A1; // Steuertaste 2 15
const int Taste_3 = A2; // Steuertaste 3 16
const int Taste_4 = A3; // Steuertaste 4 17
int Taste_Wert [4] = { 0, 0, 0, 0 }; // Variable für Tastenabfrage, als 0 vorbelegt
int entprellzeit = 100 ; // Tastenprellen abwarten in msec
const int ohne_Drueck = 500 ; // Analogwert unter 500: nichts gedrückt
const int klein_Drueck = 850 ; // Analogwert über 850: 470 Ohm gedrückt
const int gross_Drueck_u = 600 ; // Analogwert zw. 600 und 800: 4,7 K gedrückt
const int gross_Drueck_o = 800 ;
//
```

Ich habe alle Zeilen kommentiert.
Die letzten 4 Zeilen zeigen die Rückgabewerte, je nachdem welche Taste gedrückt wurde (keine, gegen 470 oder 4,7K)
Zum Tastenentprellen habe ich 100 ms vorgesehen, was in meinem Fall gut funktioniert.
4 Eingänge liefern 4 Rückgabewerte, die ich in einem Feld Taste_Wert speichere.

Die Tastenabfrage erfolgt in einer Funktion

```
//
// ----------------------------Tastenabfrage -----------------------------
//
int Druck_Taste(int dieseTaste){ // Fragt Tasten ab
int Druck_Wert = analogRead(dieseTaste); // Lese Tastennummer von Analog Port
delay (entprellzeit); // warte wegen Prellen
Druck_Wert = analogRead(dieseTaste); // frag Taste noch einmal
Taste_Wert [dieseTaste] = Druck_Wert ; // lese Wert in Array
if (Druck_Wert < ohne_Drueck) {return 0; } // keine Taste gedrückt
else if (Druck_Wert > ohne_Drueck) {return 1; } // Taste gedrückt
}
//
```

dabei kann dieseTaste 0, 1, 2, oder 3 sein (entsprechend A0 bis A3).
Die Funktion liefert 0 zurück solange keine Taste gedrückt ist.
Wenn doch (egal welche an jeweils einem AnalogPort, dann Rückgabewert 1).

In der Programmschleife wird die Funktion dann wie folgt genutzt. Da ein Motor sowohl automatisch per Programm als auch per Tastendruck gestartet werden kann, reicht eine einfache WHILE nicht aus.

```
//
// Motor 1 checken, wird über Drucktaste an A0 gestartet
//
if (Druck_Taste(Taste_1-14)==1) // nur wenn die Taste überhaupt
```

```
{ // gedrückt wurde
while (Druck_Taste(Taste_1-14)==1) // Schleife solange Taste gedrückt
{
Motor (Mot1, an , Taste_Wert[Taste_1-14] > klein_Drueck) ; // Motor 1 anschalten
} // Richtung je nach Taste an A0
Motor (Mot1, aus , HIGH) ; // Nach Loslassen der Taste
} // Motor 1 ausschalten
//
```

Die Taste A0 liefert als Integerwert 14, also muss "Taste_1" (= A0, = 14) um 14 reduziert werden und liefert "0", z.B. für den Platz im Array.
Die übergeordnete IF Abfrage stellt fest, ob überhaupt eine Taste gedrückt wurde.
Ist dies der Fall, dann läuft WHILE so lange, wie die Taste gedrückt wurde. Nach Ende der WHILE Bedingung wird der Motor definiert angehalten.
Die Motorsteuerung findet in der Prozedur Motor statt.

- erster Parameter = Motornummer (1 oder 2)
- zweiter Parameter = an oder aus (boolesche variable
- dritter Parameter = Drehrichtung, sie hängt von der Taste am Analogport ab (470 Ohm = rechtsrum, 4,7K = linksrum)

Nun kommt noch die Motorprozedur:

```
//
// --------------------------- Motor ansteuern ------------------------------
//
int Motor(int wen, boolean start, boolean wie){ // drei Parameter
// Motornummer
// wenn start HIGH lauf los
// wenn start LOW bleib stehen
// wenn wie HIGH rechts herum
// wenn wie LOW links herum
if (!start) { // soll stehen bleiben
digitalWrite(bridgePins[wen-1][0], LOW) ; // setzte H-Bridge auf Stop
digitalWrite(bridgePins[wen-1][1], LOW) ;
return wen; // gib Motornummer zurück
}
else if (start) { // soll loslaufen
digitalWrite(bridgePins[wen-1][0], wie) ; // setzte H-Bridge auf Los
digitalWrite(bridgePins[wen-1][1], !wie) ;
return wen;
}
else {return 0;}
}
//
```

Mit der Kommentierung sollte es klar sein. Die bridgePins sind die 6 Pins des Arduino, die mit der H-Bridge verbunden wurden, siehe oben.
Wenn die H-Bridge mit zweimal LOW oder HIGH angesprochen wird, bleibt der Motor stehen. Wenn sie nicht gleich sind, dann dreht der Motor rechts oder links.

Ausser bei dem Entprellen gibt es kein DELAY().

Das zeigt sich z.B. an der Motoransteuerung / Beleuchtungssteuerung etc per Programm.

Erst einmal für den Motor 1 (Drehmotor Fallerkran)

Zunächst die Definitionen:

```
//
// Motorpins generell
//
int bridgePins [2][2] = {{9, 8 }, {7, 6}} ; // Array für zwei Motor Pins,
// Motor 1: 9,8 / Motor 2: 7,6
```

```
boolean an = HIGH; // definiere für Motor Zustand an
boolean aus = LOW; // definiere für Motor Zustand aus
boolean links = LOW ; // definiere für Motor Linkslauf
boolean rechts = HIGH ; // definiere für MotorRechtslauf
//
// Motor 1
//
const int PWM_1 = 10; // PWM für Motor 1
const int Mot1 = 1; // Motor 1
unsigned long previousMillis_Mot1 = 0; // Zeitmerker Motor 1
unsigned long wann_Mot1[2] = { 60000,8000} ; // Startabstand 60000, Laufzeit Motor
1 8000
boolean next_Mot1 = rechts ; // nächste Drehrichtung vorbelegen
int status_Mot1 = 0 ; // 0 = soll laufen, 1 = soll stoppen
//
```

Durch Versuche wurde der Parameter für die Geschwindigkeit ermittelt. Der Wert wird in der Loop gesetzt:
Ausserdem wird die abgelaufene Zeit abgefragt = currentMillis

```
//
void loop() {
//
unsigned long currentMillis = millis(); // abgelaufene Zeit abfragen
//
analogWrite(PWM_1, 40); // Motor 1 speed setzen
analogWrite(PWM_2, 80); // Motor 2 speed setzen
//
```

In der Loop wird u.a. die Zeit abgefragt, wann der Motor 1 an der Reihe ist. In dem Feld wann_Mot1 stehen die Werte für die Startzeit und die Laufzeit des Motors

```
//
// Motor 1: Drehmotor Fallerkran
//
if ((currentMillis - previousMillis_Mot1 >= wann_Mot1[status_Mot1]) && status_Mot1
== 0)
{ // soll loslaufen
previousMillis_Mot1 = currentMillis; // Merke Zeitpunkt für Motor 1
Motor (Mot1, an , next_Mot1) ; // Motor an
next_Mot1 = !next_Mot1 ; // Motor Richtung umkehren
status_Mot1 = 1-status_Mot1 ; // ändern auf stehenbleiben
}
//
if ((currentMillis - previousMillis_Mot1 >= wann_Mot1[status_Mot1]) && status_Mot1
== 1)
{ // soll stehen bleiben
previousMillis_Mot1 = currentMillis; // Merke Zeitpunkt für Motor 2
status_Mot1 = 1-status_Mot1; // ändern auf loslaufen
Motor (Mot1, aus, HIGH) ; // Motor aus
previousMillis_Kabine = currentMillis; // merken für Kabinenlicht
previousMillis_deckPin1 = currentMillis; // merken für Deckenlicht
}
//
```

Eine ähnliche Abfrage gibt es für das Deckenlicht und das Blinklicht.
Das Kabinenlicht soll erst angehen, danach mit einiger Verzögerung das Blinklicht und der Drehmotor. Es soll blinken, solange der Motor sich bewegt und der Kran sich dreht.
Wenn er stehenbleibt, soll das Kabinenlicht noch kurz an bleiben und dann erlöschen.

Nun folgt der komplette Programmcode im derzeitigen Status.
Er enthält noch nicht
- automatische Ansteuerung des Märklin Kranes (Drehmotor) - Relais 1 und Relais 2
- letztendliche Drehbewegung des zweiten Motors für den Wasserkran
- weitere LED über D0 und D1

- Ansteuerung des Relais 3

Wenn das fertig ist aktualisiere ich den Code.

Hinweis: Die Schreibweise der Parameter etc mag nicht jedem gefallen. "*Heute macht man das eventuell anders*".
Ich bitte daher zu akzeptieren, dass ich mein erstes Programm in ALGOL 60 im Jahre des Herrn 1968 geschrieben habe und mir seitdem in FORTRAN, BASIC, TURBOPASCAL, dBASE II+, CLIPPER und C++ meinen eigenen Stil angewöhnt habe, den ich nicht zu ändern gedenke. 😎😆

Ansonsten sind Kommentare und Anregungen sehr willkommen. Spezielle und individuelle Fragen gern auch per PN.
Nach dem Gesetz der größten Gemeinheit waren natürlich noch Fehler im Programm. Nicht in der Kernfunktion des Fallerkrans, aber in den weiteren Funktionen (Relais für die Programmsteuerung der Relais für den Märklin Kran, Abschaltung des zweiten Motors).
Das habe ich korrigiert und auch gleich eingebaut, dass nach jedem Halt des Fallerkrans die Startzeit für die nächste Bewegung per RANDOM zufällig neu gesetzt wird. Die Grenzwerte sind

wann_Mot1[0] = random (60000, 600000) ; // Einschaltzeit zufällig neu setzen

im Moment 60 Sekunden = 1 Minute sowie 600 Sekunden = 10 Minuten. Diese Werte können nach Lust und Laune selbst gesetzt werden.
Dieselbe Funktion ist bereits vorgesehen für den Motor 2 und den Märklin Kran über die Relais Rel 1 und Rel 2, aber im Moment noch auskommentiert. Da ich noch weitere Funktionen teste mag ich nicht 10 Minuten warten

CODE:

```
//
// Ansteuerung Faller Kran
/*
 * Automatischer Betrieb zur Drehbewegung bis 270 Grad rechts und links.
 * LED in Kanzel und Führerhaus
 * Je zwei Taster auf A0 bis A3
 * A0 für direkte manuelle Drehung rechts links
 * A1 für zweiten Motor
 * A2 für Relais 1 & Relais 2 (Märklin Kran)
 * A3 zur freien Verfügung
 */

// Wolfgang aka Running_Wolf
// 2020-07-24
//
// LED
const int     Kabine       = 13;               // Führer Kabine
const int     verzKabine   = 3000;             // Einchaltverzug vor Drehstart
unsigned long previousMillis_Kabine = 0;       // Zeitmerker Kabinenlicht
const int     rotPin1      = 12;               // rotes Blinklicht
unsigned long previousMillis_rotPin1 = 0;      // Zeitmerker Blinklicht
unsigned long wann_rotPin1 = 200 ;             // rotes Blinklicht
const int     deckPin1     = 11;               // weisses Licht innen
const int     verzDeck     = 5000;             // Einchaltverzug vor Drehstart
unsigned long previousMillis_deckPin1 = 0;     // Zeitmerker Deckenlicht
//
// Motorpins generell
//
```

```
int bridgePins [2][2] = {{9, 8 }, {7, 6}} ;     // Array für zwei Motor Pins,
                                                // Motor 1: 9,8 / Motor 2: 7,6
boolean an = HIGH;                              // definiere für Motor Zustand an
boolean aus = LOW;                              // definiere für Motor Zustand aus
boolean links = LOW ;                           // definiere für Motor Linkslauf
boolean rechts = HIGH ;                         // definiere für MotorvRechtslauf
//
// Motor 1
//
const int     PWM_1 = 10;                       // PWM für Motor 1
const int     Mot1 = 1;                         // Motor 1
unsigned long previousMillis_Mot1   = 0;        // Zeitmerker Motor 1
unsigned long wann_Mot1[2] = { 30000,6000} ;    // Startabstand 60000, Laufzeit
                                                // Motor 1 8000
boolean       next_Mot1   = rechts ;            // nächste Drehrichtung vorbelegen
int           status_Mot1 = 0 ;                 // 0 = soll laufen, 1 = soll stoppen
//
// Motor 2
//
const int     PWM_2 = 5;                        // PWM für Motor 2
const int     Mot2 = 2;                         // Motor 2
unsigned long previousMillis_Mot2   = 0;        // Zeitmerker Motor 2
unsigned long wann_Mot2[2] = { 10000,8000} ;    // Startabstand 30000, Laufzeit
                                                // Motor 2 8000
boolean       next_Mot2   = rechts ;            // nächste Drehrichtung vorbelegen
int           status_Mot2 = 0 ;                 // 0 = soll laufen, 1 = soll
                                                // stoppen
//
// Motor Märklin Kran Drehen
//
unsigned long previousMillis_Mark   = 0;        // Zeitmerker Märklin Kran
unsigned long wann_Mark[2] = { 60000,8000} ;    // Startabstand 30000, Laufzeit
                                                // Motor 2 8000
boolean       next_Mark = rechts ;              // nächste Drehrichtung vorbelegen
int           status_Mark = 0 ;                 // 0 = soll laufen, 1 = soll
                                                 // stoppen
//
// Relaisausgänge Digitalpins für Relais
//
const int Rel_1  = 4;                           // Märklin Kran rechts
const int Rel_2  = 3;                           // Märklin Kran links
const int Rel_3  = 2;                           // noch frei
//
// Analogeingänge
// An jedem Analogeingang sind zwei Taster
// Pull-up Widerstand eingeschaltet
// 10 K Widerstand gegen GND
// Taster a über 470 Ohm gegen Vcc
// Taster b über 4,7 K gegen Vcc
//
const int Taste_1 = A0;                         // Steuertaste 1 auf Analog 0 liefert integer
14
const int Taste_2 = A1;                         // Steuertaste 2 15
const int Taste_3 = A2;                         // Steuertaste 3 16
const int Taste_4 = A3;                         // Steuertaste 4 17
int       Taste_Wert [4] = { 0, 0, 0, 0 };      // Variable für Tastenabfrage,
                                                // als 0 vorbelegt
int       entprellzeit   = 100 ;                // Tastenprellen abwarten in msec
const int ohne_Drueck    = 500 ;                // Analogwert unter 500: nichts gedrückt
const int klein_Drueck   = 850 ;                // Analogwert über 850: 470 Ohm gedrückt
const int gross_Drueck_u = 600 ;                // Analogwert zw. 600 und 800: 4,7 K gedrückt
const int gross_Drueck_o = 800 ;
//
const int Pin_0   = 0 ;                         // Noch frei, als Test Pin genutzt
const int Pin_1   = 1 ;                         // Noch frei, als Test Pin genutzt
//
void setup()
```

```
{
//
// Tasten / Relais einstellen auf INPUT / OUTPUT
//
pinMode(Pin_0  ,  OUTPUT);                    // Noch frei, als Test Pin genutzt
pinMode(Pin_1  ,  OUTPUT);                    // Noch frei, als Test Pin genutzt
pinMode(Rel_1  ,  OUTPUT);                    // Märklin Kran rechts
pinMode(Rel_2  ,  OUTPUT);                    // Märklin Kran linkss
pinMode(Rel_3  ,  OUTPUT);                    // noch frei
pinMode(rotPin1,  OUTPUT);                    // rotes Blinklicht
pinMode(deckPin1, OUTPUT);                    // weisses Deckenlicht
pinMode(Kabine,   OUTPUT);                    // Führerkabine
//
// Motoranschlusspins setzen
//
pinMode(PWM_1,    OUTPUT);                    // PWM Anschluss Motor 1
pinMode(PWM_2,    OUTPUT);                    // PWM Anschluss Motor 2
pinMode(bridgePins[0][0],      OUTPUT);       // Steuerpin Motor 1
pinMode(bridgePins[0][1],      OUTPUT);       // Steuerpin Motor 1
pinMode(bridgePins[1][0],      OUTPUT);       // Steuerpin Motor 2
pinMode(bridgePins[1][1],      OUTPUT);       // Steuerpin Motor 2
//
// Interner Pullup Widerstand einschalten
//
pinMode(Taste 1, INPUT PULLUP) ;              // Steuertaste 1 auf Analog 0 liefert integer
14
pinMode(Taste_2, INPUT_PULLUP) ;
pinMode(Taste_3, INPUT_PULLUP) ;
pinMode(Taste_4, INPUT_PULLUP) ;
}
//
// --------------------------------- LOOP geht los ----------------------------
//
void loop()  {
//
unsigned long currentMillis = millis();                 // abgelaufene Zeit abfragen
//
   analogWrite(PWM_1, 40);                              // Motor 1  speed
   analogWrite(PWM_2, 80);                              // Motor 2  speed
//
  digitalWrite (Pin_0, HIGH) ;                          // Test Pin 0 aus
  digitalWrite (Pin_1, LOW) ;                           // Test Pin 1 aus
//
// Motor 1 checken, wird über Drucktaste an A0 manuell gestartet
//
   if (Druck_Taste(Taste_1-14)==1)                      // nur wenn die Taste überhaupt
      {                                                 // gedrückt wurde
      while (Druck_Taste(Taste_1-14)==1)                // Schleife solange Taste gedrückt
         {
         Motor (Mot1, an , Taste_Wert[Taste_1-14] > klein_Drueck) ;  // Motor 1 anschalten
         }                                              // Richtung je nach Taste an A0
         Motor (Mot1, aus , HIGH) ;                     // Nach Loslassen der Taste
      }                                                 // Motor 1 ausschalten
//
// Motor 2 manuell drehen über A1
//
   if (Druck_Taste(Taste_2-14)==1)                      // nur wenn die Taste überhaupt
      {                                                 // gedrückt wurde
      while (Druck_Taste(Taste_2-14)==1)                // Schleife solange Taste gedrückt
         {
         Motor (Mot2, an , Taste_Wert[Taste_2-14] > klein_Drueck) ;  // Motor 2 anschalten
         }                                              // Richtung je nach Taste an A1
         Motor (Mot2, aus , HIGH) ;                     // Nach Loslassen der Taste
      }                                                 // Motor 2 ausschalten
//
// Relais 1 & 2 manuell über A2 / Taste_3
//
```

```
   if (Druck_Taste(Taste_3-14)==1)                        // nur wenn die Taste überhaupt
      {                                                   // gedrückt wurde
      while (Druck_Taste(Taste_3-14)==1)                  // Schleife solange Taste gedrückt
         {
            if (Taste_Wert[Taste_3-14] > klein_Drueck)    // Taste
            {
               digitalWrite(Rel_1, HIGH);                // Relais 1 ein
            }
            else
            {
               digitalWrite(Rel_2, HIGH);                // Relais 2 ein
            }                                             // Ende WHILE
         }                                                // Relais 1 & 2 ausschalten
            digitalWrite(Rel_1, LOW);                     // Nach Loslassen der Taste ...
            digitalWrite(Rel_2, LOW);                     // ... Rel 1 und Rel 2 aus
      }
//
// Relais 3 manuell über A3 unten = Taste_4
//
   if (Druck_Taste(Taste_4-14)==1)                        // nur wenn die Taste überhaupt
      {                                                   // gedrückt wurde
      while (Druck_Taste(Taste_4-14)==1 && (Taste_Wert[Taste_4-14] > klein_Drueck))
                                                          // Schleife solange Taste gedrückt
         {
         digitalWrite(Rel_3, HIGH);                       // Relais 3 ein halten
         }                                                // Ende WHILE
         digitalWrite(Rel_3, LOW);                        // Nach Loslassen der Taste ...
      }                                                   // ... Relais 3 ausschalten
//
//
// Deckenlicht Fallerkran
//
  if ((currentMillis-previousMillis_Mot1 >= wann_Mot1[status_Mot1]-verzDeck) &&
status_Mot1==0)
  {
         previousMillis_deckPin1 = currentMillis;         // Merke Zeitpunkt für Deckenlicht
         digitalWrite(deckPin1, HIGH) ;                   // Deckenlicht an
  }
//
  if (currentMillis - previousMillis_deckPin1 >= verzDeck && status_Mot1 == 0)
  {
         previousMillis_deckPin1 = currentMillis;         // Merke Zeitpunkt für Deckenlicht
         digitalWrite(deckPin1, LOW) ;                    // Deckenlicht aus
  }
//
// Kabinenlicht Fallerkran
//
  if ((currentMillis-previousMillis Mot1 >= wann Mot1[status Mot1]-verzKabine) &&
status_Mot1==0)
  {
         previousMillis_Kabine = currentMillis;           // Merke Zeitpunkt für Kabinenlicht
         digitalWrite(Kabine, HIGH) ;                     // Kabinenlicht an
  }
//
  if (currentMillis - previousMillis_Kabine >= verzKabine && status_Mot1 == 0)
  {
         previousMillis_Kabine = currentMillis;           // Merke Zeitpunkt für Kabinenlicht
         digitalWrite(Kabine, LOW) ;                      // Kabinenlicht aus
  }
//
// Motor 1: Drehmotor Fallerkran
//
  if ((currentMillis - previousMillis_Mot1 >= wann_Mot1[status_Mot1]) && status_Mot1 == 0)
  {                                                       // soll loslaufen
         previousMillis_Mot1 = currentMillis;             // Merke Zeitpunkt für Motor 1
         Motor (Mot1, an , next_Mot1) ;                   // Motor an
         next_Mot1 = !next_Mot1 ;                         // Motor Richtung umkehren
         status_Mot1 = 1-status_Mot1 ;                    // ändern auf stehenbleiben
```

```
  }
//
  if ((currentMillis - previousMillis_Mot1 >= wann_Mot1[status_Mot1]) && status_Mot1 == 1)
  {                                                     // soll stehen bleiben
         previousMillis_Mot1 = currentMillis;           // Merke Zeitpunkt für Motor 2
         status_Mot1 = 1-status_Mot1;                   // ändern auf loslaufen
         Motor (Mot1, aus, HIGH) ;                      // Motor aus
         wann_Mot1[0] = random (60000, 600000) ;        // Einschaltzeit zufällig neu setzen
         previousMillis_Kabine = currentMillis;         // merken für Kabinenlicht
         previousMillis_deckPin1 = currentMillis;       // merken für Deckenlicht
  }
//
// Blinklicht Bewegung Faller Kran
//
   if (currentMillis - previousMillis_rotPin1 >= wann_rotPin1)
      {                                                 // rotes Blinklicht,
                                                        // wenn Kran bewegt
      previousMillis_rotPin1 = currentMillis;           // merken für Blinklicht
      if (status_Mot1==1) {
         digitalWrite(rotPin1, !digitalRead(rotPin1));
         }                                              // Blinklicht an/aus
      else {
         digitalWrite(rotPin1, LOW); }                  // Blinklicht aus
      }
//
// Motor 2: Drehmotor
//
  if ((currentMillis - previousMillis_Mot2 >= wann_Mot2[status_Mot2]) && status_Mot2 == 0)
  {                                                     // Motor 2 soll loslaufen
         previousMillis_Mot2 = currentMillis;           // Merke Zeitpunkt für Motor 2
         Motor (Mot2, an , next_Mot2) ;                 // Motor an
         next_Mot2 = !next_Mot2 ;                       // Motor Richtung umkehren
         status_Mot2 = 1-status_Mot2 ;                  // ändern auf stehenbleiben
  }
//
  if ((currentMillis - previousMillis_Mot2 >= wann_Mot2[status_Mot2]) && status_Mot2 == 1)
  {                                                     // Motor 2 soll stehen bleiben
         previousMillis_Mot2 = currentMillis;           // Merke Zeitpunkt für Motor 2
         status_Mot2 = 1-status_Mot2;                   // ändern auf loslaufen
         Motor (Mot2, aus, HIGH) ;                      // Motor aus
//       wann_Mot2[0] = random (60000, 600000) ;          // Einschaltzeit zufällig neu
setzen
  }
//
// Märklin Kran: Drehmotor
//
  if ((currentMillis - previousMillis_Mark >= wann_Mark[status_Mark]) && status_Mark == 0)
  {                                                     // soll loslaufen
         previousMillis_Mark = currentMillis;           // Merke Zeitpunkt für Drehkran
         digitalWrite(Rel_1, next_Mark);                // Relais 1 an   für rechts
         digitalWrite(Rel_2, !next_Mark);               // Relais 2 aus und umgekehrt
         next_Mark   = !next_Mark ;                     // Motor Richtung umkehren
         status_Mark = 1-status_Mark ;                  // ändern auf stehenbleiben
  }
//
  if ((currentMillis - previousMillis_Mark >= wann_Mark[status_Mark]) && status_Mark == 1)
  {                                                     // soll stehen bleiben
         previousMillis_Mark = currentMillis;           // Merke Zeitpunkt für Drehkran
         status_Mark = 1-status_Mark;                   // ändern auf loslaufen
         digitalWrite(Rel_1,  LOW);                     // Relais 1 aus
         digitalWrite(Rel_2,  LOW);                     // Relais 2 aus
//       wann_Mark[0] = random (60000, 600000) ;        // Einschaltzeit zufällig neu setzen
  }
//
}
//
// --------------------------------- LOOP zu Ende ----------------------------
```

```
//
// --------------------------- Motor ansteuern -------------------------------
//
int Motor(int wen, boolean start, boolean wie){          // drei Parameter
                                                         // Motornummer
                                                         // wenn start HIGH lauf los
                                                         // wenn start LOW  bleib stehen
                                                         // wenn wie HIGH rechts herum
                                                         // wenn wie LOW links herum
  if (!start) {                                          // soll stehen bleiben
    digitalWrite(bridgePins[wen-1][0], LOW) ;            // setzte H-Bridge auf Stop
    digitalWrite(bridgePins[wen-1][1], LOW) ;
    return wen;                                          // gib Motornummer zurück
  }
  else if (start) {                                      // soll loslaufen
    digitalWrite(bridgePins[wen-1][0],  wie) ;           // setzte H-Bridge auf Los
    digitalWrite(bridgePins[wen-1][1], !wie) ;
    return wen;
  }
  else {return 0;}
}
//
// ----------------------------Tastenabfrage ------------------------------
//
int Druck_Taste(int dieseTaste){                         // Fragt Tasten ab
  int Druck_Wert = analogRead(dieseTaste);               // Lese Tastennummer von Analog Port
  delay (entprellzeit);                                  // warte wegen Prellen
  Druck_Wert = analogRead(dieseTaste);                   // frag Taste noch einmal
  Taste_Wert [dieseTaste] = Druck_Wert ;                 // lese Wert in Array
  if (Druck_Wert < ohne_Drueck)       {return 0; }       // keine Taste gedrückt
  else if (Druck_Wert > ohne_Drueck)  {return 1; }       // Taste gedrückt
}
//
```