FERNSTEUERUNG FÜR FAHRZEUGHEBEBÜHNE MIT ARDUINO UND 3D-DRUCK (FUNKSTEUERUNG ODER BLUETOOTH)

geschrieben von Peter S. | Juni 23, 2021

Zusammenfassung

Eine Fernbedienung kann das Leben leichter machen. Vor allem dann, wenn man Arbeiten alleine ohne Unterstützung tätigt. In der Hobbywerkstatt kann dies häufiger der Fall sein. Eine Hebebühne wird in der Regel über eine Steuerung an den Hubsäulen bedient – es ist nicht möglich, die Knöpfe zu betätigen und gleichzeitig die Ausrichtung am Fahrzeug zu steuern, zum Beispiel beim Ein- oder Ausbau eines Motors/Getriebes. Im Folgenden wurde eine konventionelle Fahrzeughebebühne mit einer Funkfernsteuerung auf Arduino-Basis nachgerüstet. Die Materialkosten belaufen sich auf ca. 30 €, die Installation ist zuverlässig und alltagstauglich.
Wir haben zunächst eine Version entwickelt, die auf **Funkkommunikation (RC)** basiert. Das System funktioniert prinzipiell einwandfrei, aber in unserer Werkstatt gab es Störfrequenzen, so dass wir uns schließlich für eine zuverlässigere **Bluetooth-Variante (BT)** entschieden. Beide Ansätze werden hier präsentiert.

Achtung: Für dieses Projekt sind Kenntnisse auf dem Gebiet der Elektrotechnik erforderlich, um eine fachgerechte und sichere Installation zu gewährleisten. Im schlimmsten Fall kann es zum Ausfall der Steuerung der Hebebühne kommen, diese bewegt sich unkontrolliert oder stürzt ab. Dies kann zu immensen Schäden an Personen oder Gegenständen führen. Die Autoren distanzieren sich hiermit vollständig von jeglicher Verantwortung! Handeln Sie auf eigene Verantwortung! Niemals unter hängenden Lasten aufhalten!

Einleitung

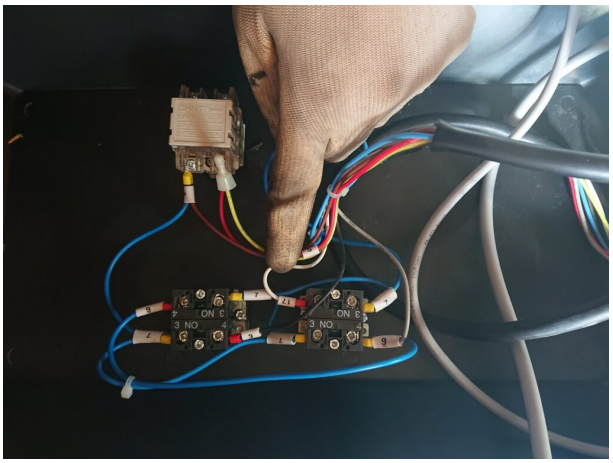
Der normale Workflow an einer Fahrzeughebebühne sieht folgendermaßen aus: Die Tragarme werden unter den Aufnahmepunkten des Fahrzeugs positioniert und die Hebebühne wird leicht angehoben. Nach einer erneuten Kontrolle an den Hebepunkten kann die Hebebühne weiter angehoben werden. Sobald die Räder des Fahrzeugs frei in der Luft sind, sollte vor dem weiteren Anheben eine letzte Kontrolle der Aufnahmepunkte durchgeführt werden. Nur so kann sichergestellt werden, dass das Fahrzeug sicher auf den Tragarmen aufliegt und weiter angehoben werden kann, ohne herunterzufallen.
Es kann lästig sein, nach dem mehrfachen Positionieren oder Überprüfen der Tragarme zum weiteren Anheben wieder zur Steuereinheit an den Hubsäulen zu gehen und dann wieder zum Fahrzeug zurückzukehren. Praktisch wäre hier eine Fernbedienung, so dass das Anheben direkt am Fahrzeug in einem Arbeitsgang erfolgen kann.

Eine weitere Situation, in der eine Fernsteuerung für die Hebebühne sinnvoll erscheint, ist der Ein- oder Ausbau von Motor- oder Getriebeeinheiten. Die Steuerung der Hebebühne direkt vor Ort spart Zeit und Nerven, und eine exakte Positionierung ist problemlos ohne die Hilfe einer zweiten Person möglich.

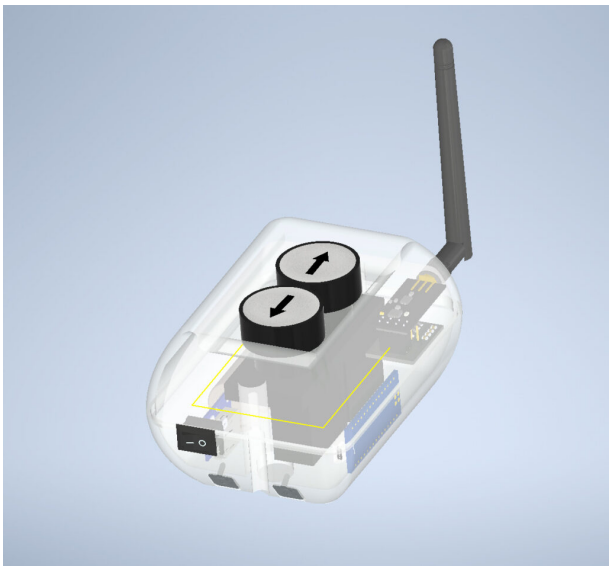
Sicherlich gibt es bereits kommerzielle Produkte, aber wir wollten dieses Projekt zum Anlass nehmen, eine einfache Arduino-basierte Lösung zu entwickeln, die sich perfekt in den Werkstattalltag integrieren lässt.

Geplant war eine Sendeeinheit als Fernsteuerung, die über robuste Tasten verfügt und auch dem harten Werkstattalltag standhält. Die Funkverbindung sollte stabil sein, sich bei einem Verbindungsverlust automatisch wieder verbinden. Der obligatorische Akkubetrieb sollte sich problemlos integrieren lassen, weshalb die Halterung gleichzeitig als einrastbare Ladeschale dient.

Hebebühnensteuerung (mit 230 VAC und 12 VDC Versorgung).



Interne Verdrahtung der Ansteuerung. (3) Auf, (17) Ab, (9) Freigabe, (7) Masse.



CAD-Modell der Fernsteuerung (RC) (Autodesk Inventor 2021)

Materialien & Methoden

Alle Gehäuseteile wurden mit PLA-Filament auf einem 3D-Drucker gedruckt. Die elektronischen Komponenten im Inneren der Sendeeinheit wurden mit 0,5 m² Litzenkabeln verlötet. Für die Empfängereinheit wurden gecrimpte Dupont-Stecker mit 2,64 mm Durchmesser (auch als “Arduino-Kabel” bekannt) als Stecker verwendet. Der Quellcode wurde über die USB-Schnittstelle auf den Mikroprozessor geflasht.

Komponenten

- Arduino Nano (x2)
- Relais 5V High trigger (x3)
- Drucktaster mit Pfeil (x2)
- Steuereinheit für Drucktaster (x2)
- Step Down Konverter
- 5 V Battery PCB TC4056A/TP4056
- 3,7 V 18650 Li-Ion Battery
- Wago 222-415 (x3)
- Ladekontakte AA (2×2)
- Neodym Magnete (2-4)
- 2 Kunststoff-/Blechschrauben

RC-Version:

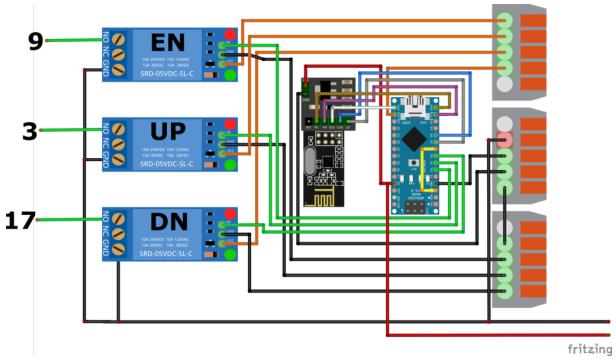
- NRF24L01 PA (Sender)
- DIP24L01 (Empfänger)
- Erweiterungsboard (x2)

BT-Version:

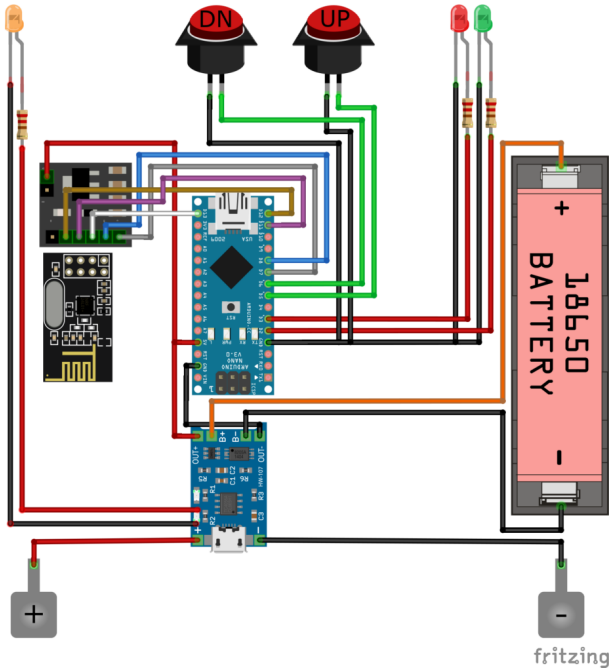
- Bluetooth-Modul HC-05 (Sender)
- Bluetooth-Modul HC-06 (Empfänger)
- Alternativ: HC-05 (x2)

Software

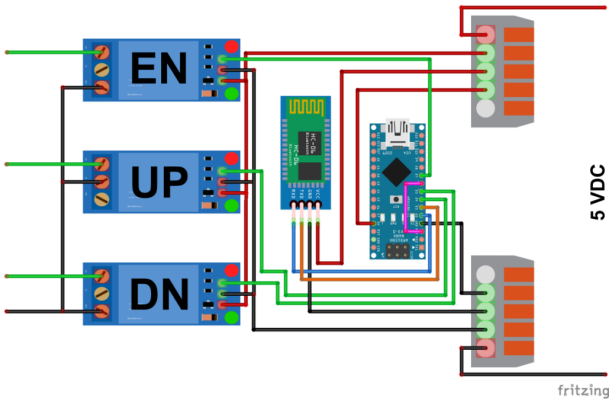
- Autodesk Inventor Professional 2019 (Build: 265, Release: 2019.2)
- Fritzing beta (Ver. 0.9.4)
- Arduino IDE 1.8.10 (Board: Arduino Nano, Programmer: AVRISP mkII, Prozessor: ATmega328P (Old Bootloader))



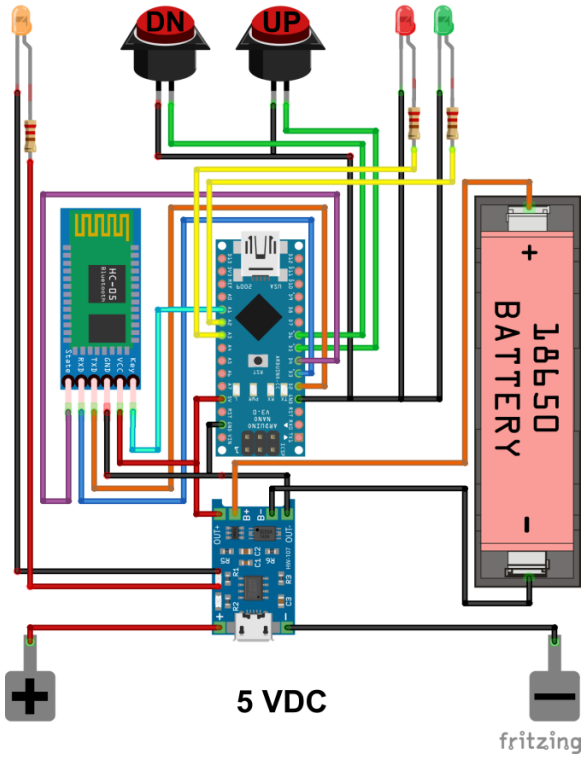
Anschlussplan Empfänger (RC)



Anschlussplan Fernbedienung mit Status-LEDs und Powerbank-Batterie (RC).
RC: Schaltpläne der Fernbedienung und der Empfängereinheit. Rot: 5 V, Orange: 3,3 V, Grün: Signal, Schwarz: Masse.



Anschlussplan Empfänger (BT)



Anschlussplan Fernbedienung mit Status-LEDs und Powerbank-Batterie (BT).
BT: Schaltpläne von Fernbedienung und Empfängereinheit. Rot: 5 V, Orange: 3,3 V, Grün: Signal, Schwarz: Masse.

Quellcode

Die Tasten der Fernbedienung senden ein bestimmtes Signal an die Empfängereinheit, das entweder das Auf- oder das Abwärtsrelais aktiviert. Das “Freigabe”-Relais ist jedoch immer aktiviert, wodurch die Sicherheitsstifte der Hebebühne bewegt werden, damit eine Bewegung überhaupt möglich ist. Aus Sicherheitsgründen wird dieses Relais nicht überbrückt, sondern aktiviert, wenn der Betrieb erwünscht ist. Fällt die Verbindung zur Empfangseinheit aus, schalten die Relais ab und die Hebebühne bleibt stehen.
Auf der Sendeeinheit werden zwei LEDs per Code gesteuert: Die grüne LED zeigt den allgemeinen Betriebszustand des Mikrocontrollers an. Die rote LED zeigt den Verbindungsstatus mit der Empfängereinheit an. Wenn sie nicht leuchtet, ist die Verbindung hergestellt und das Gerät betriebsbereit. Wenn die rote LED schnell blinkt, kann keine Verbindung zum Empfänger hergestellt werden. Während des Tastendrucks am Sender zeigt ein Dauerleuchten in der Länge des Tastendrucks die erfolgreiche Übertragung des Signals an.

Die funkgesteuerte Variante (RC) enthält zusätzlich einen Code-Teil, der im Falle einer gestörten Funkverbindung das Empfängermodul über den Reset-Pin periodisch neu startet, um einen neuen Verbindungsversuch zu unternehmen. Um das Debugging der Verbindung zu erleichtern, enthält der Code auch Meldungen, die übertragen werden und über den Serial Monitor abgerufen werden können.

Libraries

- RF24.h
- printf.h
- SPI.h
- SoftwareSerial.h

Code Sendereinheit (RC)

```
#include <SPI.h>
#include "printf.h"
#include "RF24.h"
```

```
#define TURNOFFDELAY 10000 /* in ms */

// Pins
#define LED_PIN_POWER 2
#define LED_PIN 3
#define BUTTONUP_PIN 5
#define BUTTONDOWN_PIN 6
#define CE_PIN 7
#define CSN_PIN 8

// Messages
#define UP    'U'
#define DOWN  'D'
#define STOP  'S'
#define R_PING  'I'
#define R_PONG  'O'

#define PIPE 0 //0x42216836aa

// instantiate an object for the nRF24L01 transceiver
RF24 radio(CE_PIN, CSN_PIN);

uint8_t address[][6] = {"1Node", "2Node"};
bool radioNumber = 1; // 0 uses address[0] to transmit, 1 uses address[1] to transmit
bool role = false;  // true = TX role, false = RX role

uint8_t message = STOP;

bool report = false,
    error = false;

unsigned long turnOffTime = 10000;

void setup() {
  // intit pins
//  pinMode(LDO_PIN, OUTPUT);
  pinMode(LED_PIN, OUTPUT);
  pinMode(LED_PIN_POWER, OUTPUT);
  pinMode(BUTTONUP_PIN, INPUT_PULLUP);
  pinMode(BUTTONDOWN_PIN, INPUT_PULLUP);
  digitalWrite(LED_PIN, false);
//  digitalWrite(LDO_PIN, false);


  while (!radio.begin()) { // radio is not responding
    for (int i = 0; i<5; i++) { // blink slowly for 5s
      digitalWrite(LED_PIN, true);
      delay(500);
      digitalWrite(LED_PIN, false);
      delay(500);
    }
//    digitalWrite(LDO_PIN, true);
//    delay(50);
//    digitalWrite(LDO_PIN, false);
//    for(;;);
  }

  // Set the PA Level low to try preventing power supply related problems
  // because these examples are likely run with nodes in close proximity to
  // each other.
  radio.setPALevel(RF24_PA_HIGH);  // RF24_PA_MAX is default. RF24_PA_LOW
```

```
  // save on transmission time by setting the radio to only transmit the
  // number of bytes we need to transmit a uint8_t
  radio.setPayloadSize(sizeof(message)); // uint8_t datatype occupies 1 byte

  // set the TX address of the RX node into the TX pipe
  radio.openWritingPipe(address[radioNumber]);     // always uses pipe 0

  // set the RX address of the TX node into a RX pipe
  radio.openReadingPipe(1, address[!radioNumber]); // using pipe 1

  radio.stopListening();  // put radio in TX mode

}

void loop() {
  bool buttonUp = !digitalRead(BUTTONUP_PIN);
  bool buttonDown = !digitalRead(BUTTONDOWN_PIN);
  digitalWrite(LED_PIN_POWER, HIGH);

  if (buttonUp == buttonDown == true){ // Error: Both Buttons Pressed
    message = STOP;
    error = true;
    digitalWrite(LED_PIN, false);
  } else if (buttonUp) {
    message = UP;
    digitalWrite(LED_PIN, true);
    turnOffTime = millis() + TURNOFFDELAY;
  } else if (buttonDown) {
    message = DOWN;
    digitalWrite(LED_PIN, true);
    turnOffTime = millis() + TURNOFFDELAY;
  } else if (buttonUp == buttonDown == false) {
    message = STOP;
    error = false;
    digitalWrite(LED_PIN, false);
  }

  report = radio.write(&message, sizeof(uint8_t));

  if (!report) {  // Transmission failed
    for (int i = 0; i<5; i++) { // blink fast for 0.5s
      digitalWrite(LED_PIN, true);
      delay(50);
      digitalWrite(LED_PIN, false);
      delay(50);
    }
  }
}
```

Code Sendereinheit (BT)

```
#include <SoftwareSerial.h>

// Pins
#define POWER_LED A2
#define LED_PIN A3
#define BUTTONUP_PIN 5
#define BUTTONDOWN_PIN 6
#define RXPIN 3
#define TXPIN 2
#define STATEPIN 4
#define ENABLEPIN A1
```

```
// Messages
#define UP    'U'
#define DOWN  'D'
#define STOP  'S'

#define HBBBLUETOOTHADDR 20:17:03:15:03:34
#define BINDSTR "AT+BIND=2017,03,150334"

SoftwareSerial blueSerial(RXPIN, TXPIN); // RX, TX

char message = STOP;

bool ledState = false;

void setup() {
    // intit pins
  pinMode(POWER_LED, OUTPUT);
  pinMode(LED_PIN, OUTPUT);
  pinMode(BUTTONUP_PIN, INPUT_PULLUP);
  pinMode(BUTTONDOWN_PIN, INPUT_PULLUP);
  pinMode(ENABLEPIN, OUTPUT);
  pinMode(STATEPIN, INPUT);
  digitalWrite(LED_PIN, false);
  digitalWrite(POWER_LED, true);
  digitalWrite(ENABLEPIN, true);

  Serial.begin(38400);
  while (!Serial) {
    // some boards need to wait to ensure access to serial over USB
  }

  // 9600
  Serial.println("Startup");

  if (false) { // configure bluetooth module
    blueSerial.begin(38400); //38400
    while (1) {

      if (Serial.available()>0)
        blueSerial.write(Serial.read());

      if (blueSerial.available()>0)
        Serial.write(blueSerial.read());
      }
    } else {
      blueSerial.begin(9600);
    }

} // setup

void loop() {
  bool buttonUp = !digitalRead(BUTTONUP_PIN);
  bool buttonDown = !digitalRead(BUTTONDOWN_PIN);

  bool statePin = digitalRead(STATEPIN);

  if (!statePin) {
    ledState = !ledState;

    digitalWrite(LED_PIN, ledState);

    Serial.println("Waiting for connection");
```

```
  } else {

    if (buttonUp == buttonDown == true){ // Error: Both Buttons Pressed
      message = STOP;
      digitalWrite(LED_PIN, false);
    } else if (buttonUp) {
      message = UP;
      digitalWrite(LED_PIN, true);
    } else if (buttonDown) {
      message = DOWN;
      digitalWrite(LED_PIN, true);
    } else if (buttonUp == buttonDown == false) {
      message = STOP;
      digitalWrite(LED_PIN, false);
    }
    Serial.print(statePin);
    Serial.print("Sending: ");
    Serial.write(message);
    Serial.println("\n");
    blueSerial.write(message);

  }
  delay(45);
} // loop
```

Code Empfänger (RC)

```
#include <SPI.h>
#include "printf.h"
#include "RF24.h"

//
#define TRANSMISSIONTIMEOUT 100 /* in ms */

#define RESETTIMEOUT 1000       /* in ms */

// Pins
#define RESET_PIN 6
#define LED_PIN 2
#define RELAYUP_PIN 4
#define RELAYDOWN_PIN 5
#define RELAYENABLE_PIN 3
#define CE_PIN 7
#define CSN_PIN 8

// Messages
#define UP    'U'
#define DOWN  'D'
#define STOP  'S'
#define R_PING  'I'
#define R_PONG  'O'

#define PIPE 0 //0x42216836aa

// instantiate an object for the nRF24L01 transceiver
RF24 radio(CE_PIN, CSN_PIN);

uint8_t address[][6] = {"1Node", "2Node"};
bool radioNumber = 0; // 0 uses address[0] to transmit, 1 uses address[1] to transmit

uint8_t message = STOP;

uint8_t pipe = 0;
```

```
bool report = false,
    error = false;

unsigned long expectedTransmissionTime = 10000,
              resetTimeout = 100000;

void setup() {
  // intit pins
  pinMode(LED_PIN, OUTPUT);
  pinMode(RELAYUP_PIN, OUTPUT);
  pinMode(RELAYDOWN_PIN, OUTPUT);
  pinMode(RELAYENABLE_PIN, OUTPUT);
  digitalWrite(LED_PIN, false);
  digitalWrite(RELAYUP_PIN, false);
  digitalWrite(RELAYDOWN_PIN, false);
  digitalWrite(RELAYENABLE_PIN, false);

  while (!radio.begin()) { // radio is not responding
    for (int i = 0; i<5; i++) { // blink slowly for 5s
      digitalWrite(LED_PIN, true);
      delay(500);
      digitalWrite(LED_PIN, false);
      delay(500);
    }
  }

  // Set the PA Level low to try preventing power supply related problems
  // because these examples are likely run with nodes in close proximity to
  // each other.
   radio.setPALevel(RF24_PA_HIGH);   // RF24_PA_MAX is default. RF24_PA_LOW RF24_PA_LOW
RF24_PA_MAX RF24_PA_MIN=0

  // save on transmission time by setting the radio to only transmit the
  // number of bytes we need to transmit a uint8_t
  radio.setPayloadSize(sizeof(message)); // uint8_t datatype occupies 1 byte

  // set the TX address of the RX node into the TX pipe
  radio.openWritingPipe(address[radioNumber]);     // always uses pipe 0

  // set the RX address of the TX node into a RX pipe
  radio.openReadingPipe(1, address[!radioNumber]); // using pipe 1

  radio.startListening();  // put radio in RX mode

}

void loop() {
  error = true;
  while(millis() < expectedTransmissionTime) {
    if (radio.available(&pipe)) {
      if (radio.getPayloadSize() > 1) {
        error = true;
        expectedTransmissionTime = millis() + TRANSMISSIONTIMEOUT;
        resetTimeout = millis() + RESETTIMEOUT;
        break;
      } else {
        radio.read(&message, 1);
        error = false;
        expectedTransmissionTime = millis() + TRANSMISSIONTIMEOUT;
        resetTimeout = millis() + RESETTIMEOUT;
        break;
      }
```

```
    }
  }

  if (millis() > resetTimeout) {
    pinMode(RESET_PIN, OUTPUT);
  }

  if (error) {
    digitalWrite(RELAYUP_PIN, false);
    digitalWrite(RELAYDOWN_PIN, false);
    digitalWrite(RELAYENABLE_PIN, false);
  } else {

    switch (message) {
      case UP:
        digitalWrite(RELAYUP_PIN, true);
        digitalWrite(RELAYDOWN_PIN, false);
        digitalWrite(RELAYENABLE_PIN, true);
      break;

      case DOWN:
        digitalWrite(RELAYUP_PIN, false);
        digitalWrite(RELAYDOWN_PIN, true);
        digitalWrite(RELAYENABLE_PIN, true);
      break;

      case STOP:
      default:
        digitalWrite(RELAYUP_PIN, false);
        digitalWrite(RELAYDOWN_PIN, false);
        digitalWrite(RELAYENABLE_PIN, false);
      break;
    }
  }
}
```

Code Empfänger (BT)

```
#include <SoftwareSerial.h>

//
#define TRANSMISSIONTIMEOUT 100 /* in ms */
#define RESETTIMEOUT 2000

// Pins
#define LED_PIN A1
#define RELAYENABLE_PIN 7
#define RELAYUP_PIN 4
#define RELAYDOWN_PIN 5
#define RESET_PIN 6

// Messages
#define UP    'U'
#define DOWN  'D'
#define STOP  'S'


SoftwareSerial blueSerial(2, 3); // RX, TX

unsigned long expectedTransmissionTime = 10000,
              resetTime = 10000;

char message = 0;
```

```
void setup() {
    // intit pins
  pinMode(LED_PIN, OUTPUT);
  pinMode(RELAYUP_PIN, OUTPUT);
  pinMode(RELAYDOWN_PIN, OUTPUT);
  pinMode(RELAYENABLE_PIN, OUTPUT);
  digitalWrite(LED_PIN, false);
  digitalWrite(RELAYUP_PIN, false);
  digitalWrite(RELAYDOWN_PIN, false);
  digitalWrite(RELAYENABLE_PIN, false);

  Serial.begin(115200);
  while (!Serial) {
    // some boards need to wait to ensure access to serial over USB
  }
  blueSerial.begin(9600);
} // setup

void loop() {

  while(millis() < expectedTransmissionTime) {
    if (blueSerial.available()) {
      Serial.print(F("Received "));
      message = blueSerial.read();
      Serial.print(message);
      Serial.println("  .");
     resetTime = millis() + RESETTIMEOUT;
     break;
    }
  }

  if (millis() >= expectedTransmissionTime) {

    digitalWrite(RELAYUP_PIN, false);
    digitalWrite(RELAYDOWN_PIN, false);
    digitalWrite(RELAYENABLE_PIN, false);
    message = STOP;
  }
    switch (message) {
      case UP:
        digitalWrite(RELAYUP_PIN, true);
        digitalWrite(RELAYDOWN_PIN, false);
        digitalWrite(RELAYENABLE_PIN, true);
        expectedTransmissionTime = millis() + TRANSMISSIONTIMEOUT;
      break;

      case DOWN:
        digitalWrite(RELAYUP_PIN, false);
        digitalWrite(RELAYDOWN_PIN, true);
        digitalWrite(RELAYENABLE_PIN, true);
        expectedTransmissionTime = millis() + TRANSMISSIONTIMEOUT;
      break;

      case STOP:
        digitalWrite(RELAYUP_PIN, false);
        digitalWrite(RELAYDOWN_PIN, false);
        digitalWrite(RELAYENABLE_PIN, false);
        expectedTransmissionTime = millis() + TRANSMISSIONTIMEOUT;
      break;
  }
} // loop
```

Ergebnisse

Empfänger

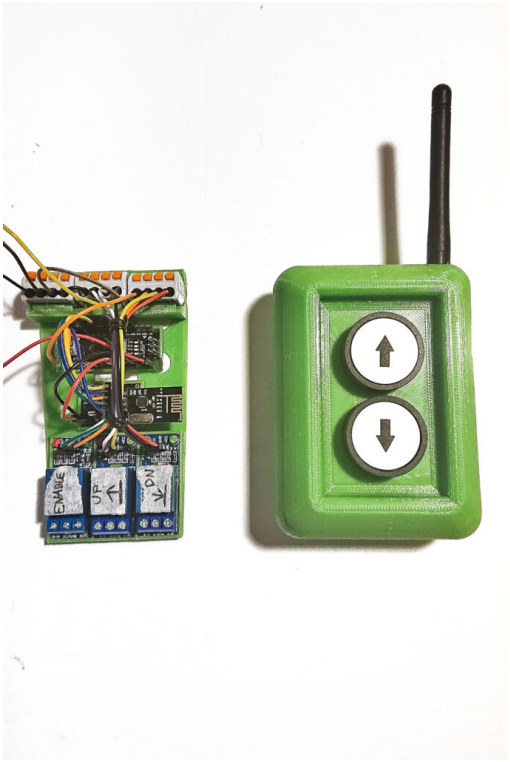
Es wurden drei Relais verwendet, die im Schaltschrank parallel zu den beiden vorhandenen Drucktastern angeschlossen wurden. Zusätzlich zu je einem Relais für die Auf- und Abwärtsfahrt gibt ein drittes Relais das Freigabesignal weiter und entriegelt den Sicherheitsmechanismus des Aufzugs. Die Relais arbeiten mit einem high-trigger, so dass bei einer Unterbrechung der Signalverbindung zur Fernsteuerung oder der Stromversorgung der Sendeeinheit keine ungewollte Bewegung der Hebebühne erfolgt und somit die Sicherheit gewährleistet ist. Zusammen mit einem Arduino Nano und dem NRF24L01 (RC)-Funkmodul oder einem HC-06 (BT)-Bluetooth-Modul wurde die Empfängereinheit an eine 5-VDC-Verteilung angeschlossen, die mit Hilfe eines Tiefsetzstellers/Step-Down (nicht abgebildet) von der 12-VDC-Spannung des Aufzugs gespeist wird. Es ist wichtig, dass der Arduino und seine Module mit 5 VDC versorgt werden. Wenn die Hebebühne keine 12 VDC liefert, sondern vielleicht nur Netzspannung (230 VAC), muss ein Netzgerät (~2 A) verwendet werden.

Sender

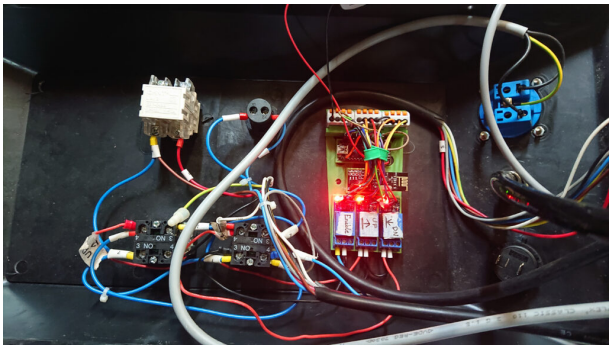
Neben einem Arduino Nano mit Funkmodul (NRF24L01) oder Bluetooth-Modul (HC-05) verfügt die Sendeeinheit über solide Knöpfe für den Werkstattalltag und einen 3,7-V-Akku mit 5-V-Lademechanismus, der Einfachheit und Zuverlässigkeit garantiert – und eine lange Laufzeit besitzt.

LEDs zur Statusanzeige komplettieren die Technik der Sendeeinheit.
Die grüne LED leuchtet, sobald der Sender mit dem Wippschalter eingeschaltet und der Arduino gestartet wird.
Die rote LED informiert über den Zustand des Sendemoduls: Solange der Taster zur Bedienung des Aufzugs gedrückt wird, leuchtet die rote LED und zeigt damit an, dass ein Signal an den Empfänger gesendet wird. Blinkt die rote LED hingegen kontinuierlich, besteht keine Verbindung zum Empfänger. In diesem Fall versucht das Funkmodul in regelmäßigen Abständen, die Verbindung wiederherzustellen.
Die gelbe LED leuchtet, sobald sich die Sendeeinheit in der Ladestation befindet und der Akku geladen wird. Als Akku und 5-V-Laderegler wurde eine handelsübliche Powerbank mit einem 3,7-V-Akku verwendet. Je nach verwendeter Ladeelektronik leuchtet die gelbe LED dauerhaft oder blinkt und schaltet sich nach Abschluss des Ladevorgangs aus.

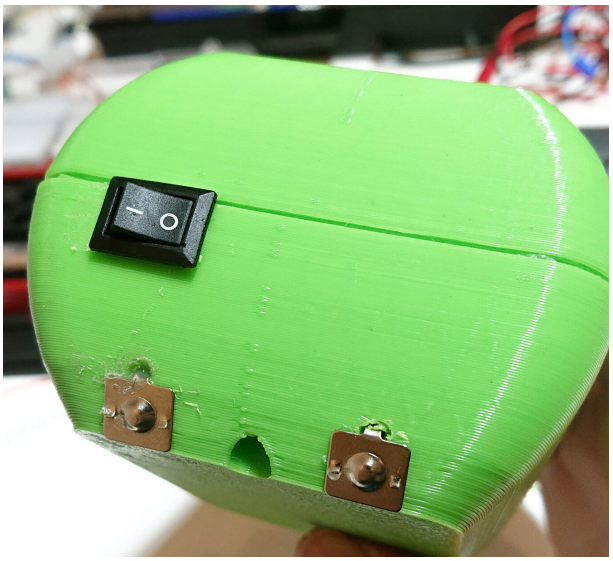
Für die Gestaltung der Ladeschale des Senders wurden Kontaktplatten für AA-Batterien verwendet und mit Heißkleber befestigt. Der Verriegelungsmechanismus wurde im CAD so angepasst, dass die Fernbedienung automatisch ausgeschaltet wird, wenn sie in die Ladeschale eingesetzt wird (Im Video dargestellt). Neodym-Magnete in der Rückwand sorgen dafür, dass die Fernbedienung an metallischen Oberflächen (auch ohne Ladeschale) haftet.



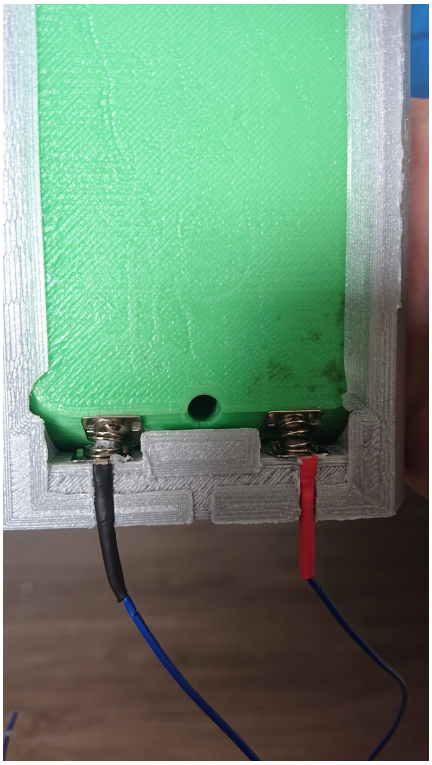
Links: Empfängereinheit (RC), Rechts: Sendereinheit (RC)



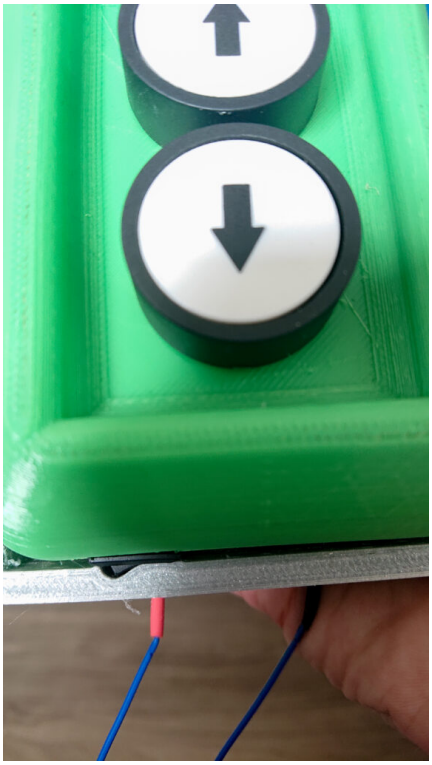
Empfängereinheit an die Elektrik der Hebebühne angeschlossen und in Betrieb.



Ladekontakte der Sendereinheit

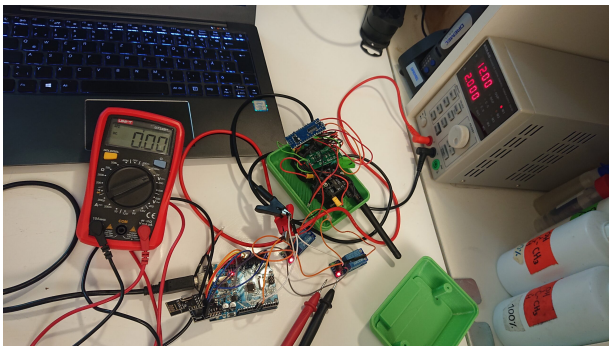


Ladekontakte der Ladeschale

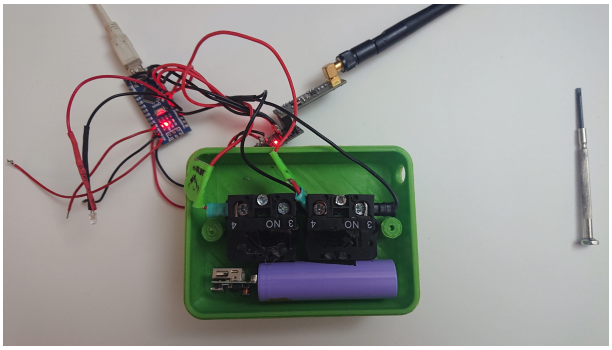


Sendereinheit in der Ladeschale: Der Schalter wird in “AUS”-Position arretiert

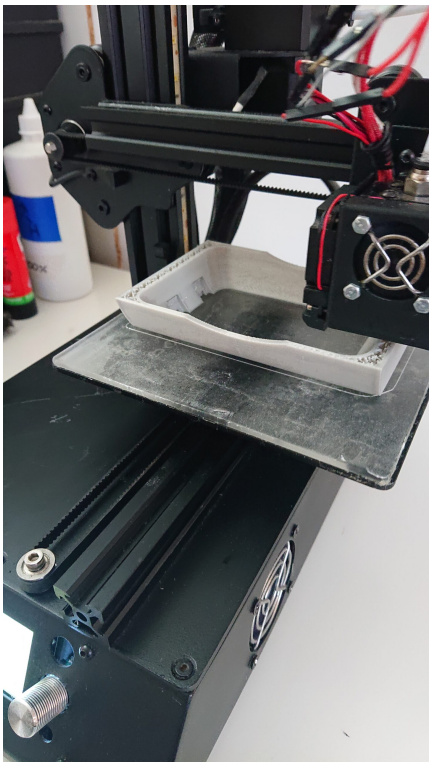
Sendereinheit: Magnete an der Rückwand halten die Einheit in der Ladeschale und auf anderen metallischen Flächen.



Impression



Impression: Der 18650 Akku passt wunderbar in das Gehäuse.



Impression

Sendereinheit: Drei LEDs an der Oberseite geben Rückschluss auf den Status. Grün: Gerätestatus, betriebsbereit. Gelb: Akkustatus, lädt. Rot: Signalübertragung, Leuchtet während Tastendruck, Blinkt bei Verbindungsabbruch

Fernbedienung (RC) in Betrieb

Fernbedienung (RC) schaltet sich beim Einsetzen in die Ladeschale ab (grüne LED) und lädt den Akku (gelbe LED). In diesem Test ebenfalls zu erkennen, wie sich ein Verbindungsabbruch äußert (rote LED)

Diskussion und Ausblick

In diesem Projekt wurde eine handelsübliche Fahrzeughebebühne mit Funkfernbedienung nachgerüstet. Die Installation gestaltete sich einfach, die Bediunung intuitiv und zuverlässig. Es wurden keine Sicherheitseinrichtungen umgangen und der Betrieb stellt bei sachgemäßer Ausführung keine Gefährdung dar. Durch die Auswahl einer von beiden Methoden (Funk vs. Bluetooth) erlaubt es sich auch unter Störfrequenzen (wie es in der Testwerkstatt der Fall ist) mit dem alternativen Funkprotokoll eine funktionsfähige und zuverlässige Lösung zu implementieren. Durch Verwendung der Funkmodule ist allerdings eine höhere Funkreichweite möglich. Beide Implementierungen wurden vorgestellt, gebaut und erfolgreich getestet. Als Erweiterung in Bezug auf Sicherheitsaspekte könnte in Betracht gezogen werden einen Notaus-Schalter an der Sendereinheit zu ergänzen – das vorgestellte System mit high-trigger Relais zeigt hierfür aber aktuell keine Notwendigkeit.