AUFGABENBLATT: TIME2RACE

TIME2RACE @ CMD+O

T-Systems on site Services GmbH

Version: 1.0

TIME2RACE @ CMD+O

T · · Systems ·

IMPRESSUM

Herausgeber	T-Systems on site Services GmbH Fasanenweg 5 70771 Leinfelden-Echterdingen

Version	Letztes Review
1.00	20.11.2020

Kurzbeschreibung
Anweisung und Hilfestellung zur Umsetzung des T-Racers

Inhaltsverzeichnis

Abbildungsverzeichnis

1 T-RACER

1.1 Was ist zu tun?

Ziel dieses Workshops ist das Entwickeln deines eigenen T-Racers, welcher per Bluetooth Low Energy steuerbar ist. Das Herzstück des T-Racers ist ein ESP32 Dev Board, welches du über die Arduino IDE programmieren kannst. Darüber steuerst du den Motor für den Antrieb, sowie den Servo Motor für die Lenkung. Für die Kommunikation zwischen App und T-Racer wird das Protokoll Bluetooth Low Energy eingesetzt.

1.2 Die Macht der Schaltung

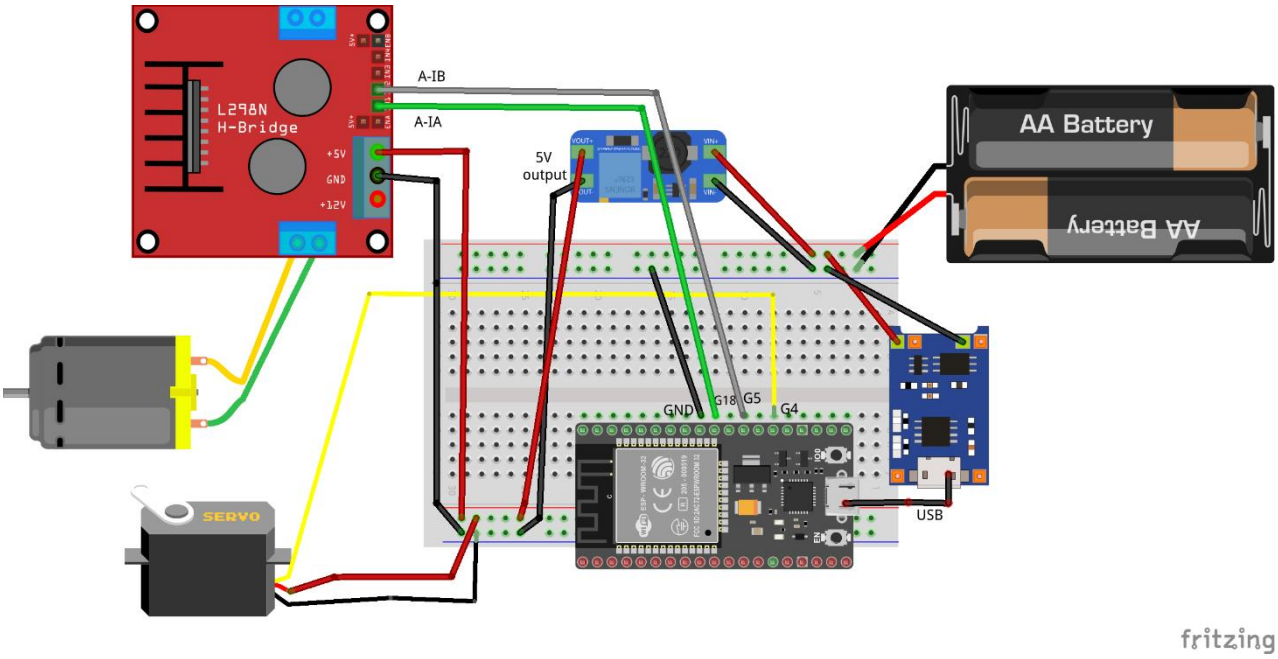


Abbildung 1: Schematischer Schaltplan

1.2.1 Hardware Komponenten

ESP 32

Der ESP32 ist eine kostengünstige und mit geringem Leistungsbedarf ausgeführte 32-Bit-Mikrocontrollerfamilie der Firma espressif. Er kommt standardmäßig mit Wifi und BLE an Bord.

Step up (MT3608)

Der MT3608 ist ein Step-Up Konverter (Auswärtswandler). Dieser wird benötigt, um die Spannung der Batterie auf konstante 5V zu regeln.

Eingangsspannung: 2V - 24V, Ausgangsspannung: bis 28V, Max Nennstrom: 2A

Power supply Modul

Wird benötigt, um die 4,5 Volt des Akkus auf konstante 5 Volt zu bringen, so kommt es nicht zu Verbindungsabbrüchen des BLE Chips. Der Spannungswandler verfügt über einen USB Output, wodurch eine einfache Stromversorgung des ESP32 ermöglicht wird.

AA-Batterien

3x 1,5V für die Stromversorgung

Servo Motor

Servo Motor für die Lenkung des T-Racers.

DC Getriebe Motor

DC Motor mit vorgeschaltener Getriebebox für den Antrieb. Durch die Getriebeübersetzung liefert der Motor genug Drehmoment, um den T-Racer zu bewegen. Max 300 rpm.

L9110 Motor controller

L9110 H-Bridge zum steuern des DC Motors. Mit dem L9110 können bis zu 2 Motoren angesteuert werden. Dieser ist erforderlich, um die Motorrichtung umkehren zu können. Mit einem einfachen MOS-FET können wir nur die Leistung ändern, aber nicht die Richtung.

Breadboard

Steckplatine für einfaches Prototyping.

WICHTIG: Wenn der T-Racer nicht verwendet wird, sollte der Akku entfernt werden!

2 SETUP

2.1 Ein Ausflug in die Bibliotheken

Um den Programmcode für unsere Christbaumkugel zu schreiben werden einige externe Bibliotheken benötigt, welche mittels „include“ eingebunden werden müssen.

Folgende Bibliotheken sind bereits durch Arduino selbst oder durch das „esp32“-Paket mitgeliefert:

- **ESP_WiFi: (Optional)** Stellt allgemeine Funktionen für WiFi-Verbindungen zu Verfügung. Wir benötigen sie aber nur um WiFi abzuschalten, damit der Stromverbrauch reduziert wird.
- **BLEDevice:** Klasse für ein BLE Device. Die Grundlage für die BLE Kommunikation.
- **BLEServer:** Klasse für einen BLE Server. Hierauf kann sich später das Smartphone verbinden.
- **BLEUtils:** Allgemeine Funktionen für BLE.
- **ESP32Servo:** Bibliothek für die Kommunikation mit dem Servo Motor

2.2 Genau die richtige Debugging App für den T-Racer – nRFConnect

Um die Bluetooth-Verbindung von Smartphone aus zu testen kann die App nRFConnect genutzt werden. Die App ist für Android sowie iOS kostenlos verfügbar:

- https://play.google.com/store/apps/details?id=no.nordicsemi.android.mcp&hl=en_US
- https://apps.apple.com/us/app/nrf-connect/id1054362403

2.3 Arduino - Was sonst?

Um das Board über euren Computer zu programmiere, müsst ihr es per Mikro-USB anschließen. Das Board wird über einen virtuellen COM Port angesprochen. Hierzu benötigt der Computer noch die passenden Treiber. Diese sind hier zu finden:

https://www.silabs.com/products/development-tools/software/usb-to-uart-bridge-vcp-drivers

Anschließend kann man sich die Arduino IDE unter https://www.arduino.cc/ für alle gängigen Betriebssysteme downloaden und installieren. Zunächst ist es wichtig, alle benötigten Boardkonfigurationen und Bibliotheken einzubinden, um den ESP32 über die IDE nutzen zu können. Dazu geht man unter „Datei -> Voreinstellungen“ und fügt eine „zusätzliche Boardverwalter-URL“ ein: *https://dl.espressif.com/dl/package_esp32_index.json*

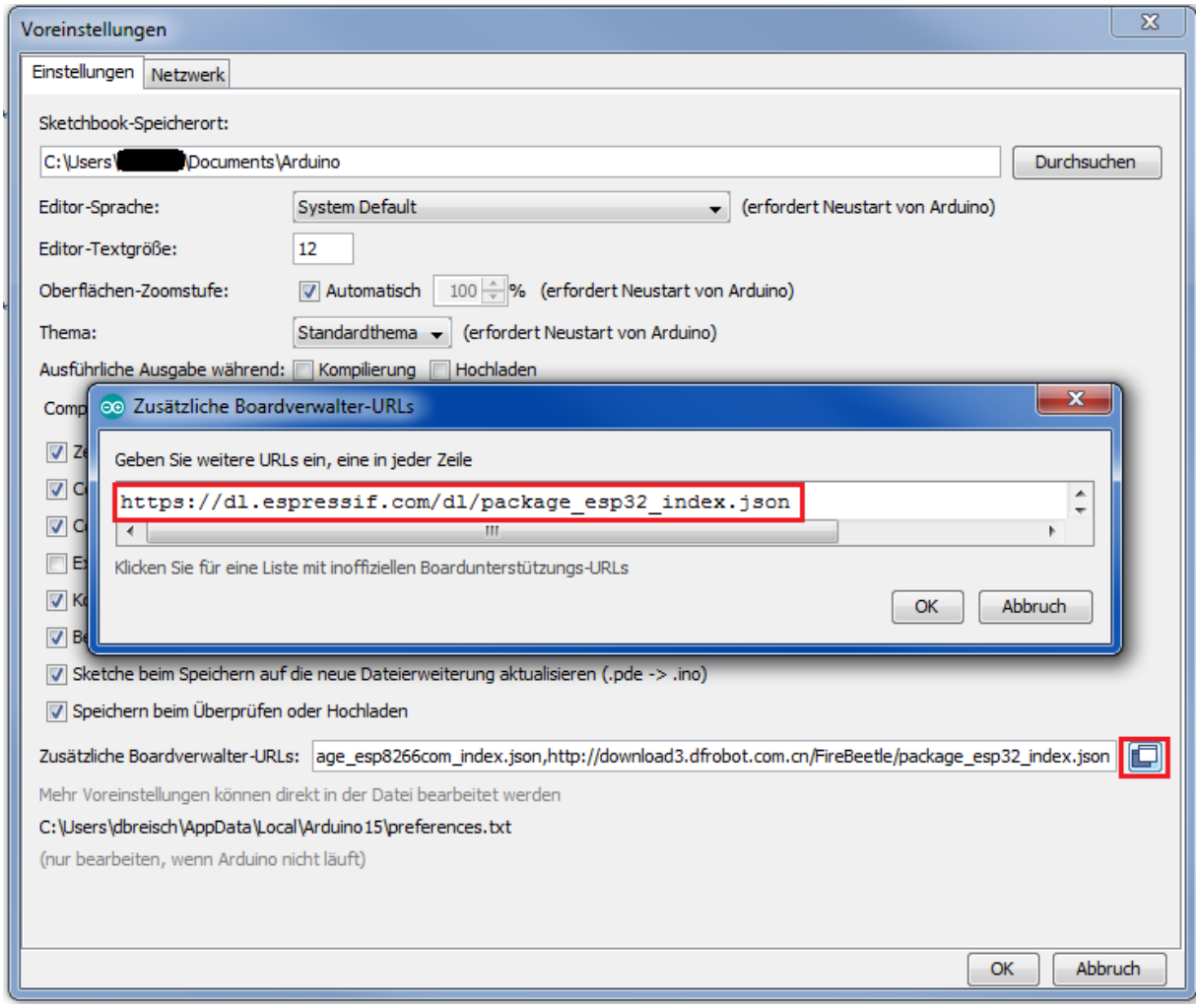


Abbildung 2: Zusätzliche Boardverwalter hinzufügen

Nun kann unter „Werkzeuge -> Board -> Boardverwalter“ das „esp32“ Paket installiert werden.

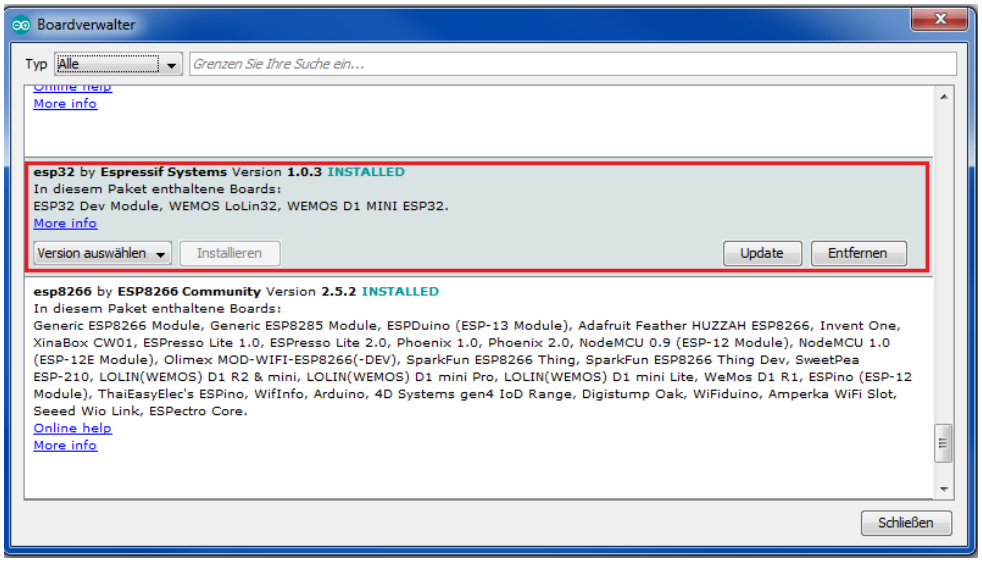


Abbildung 3: esp32 Paket installieren

Damit nun die IDE weiß für welches Board tatsächlich entwickelt wird, gibt man abschließend unter „Werkzeuge“ die korrekte Konfiguration wie in Abbildung 4 dargestellt an. Wichtig hierbei ist zudem das Auswählen des richtigen virtuelle COM Ports, an dem das Board angeschlossen ist.

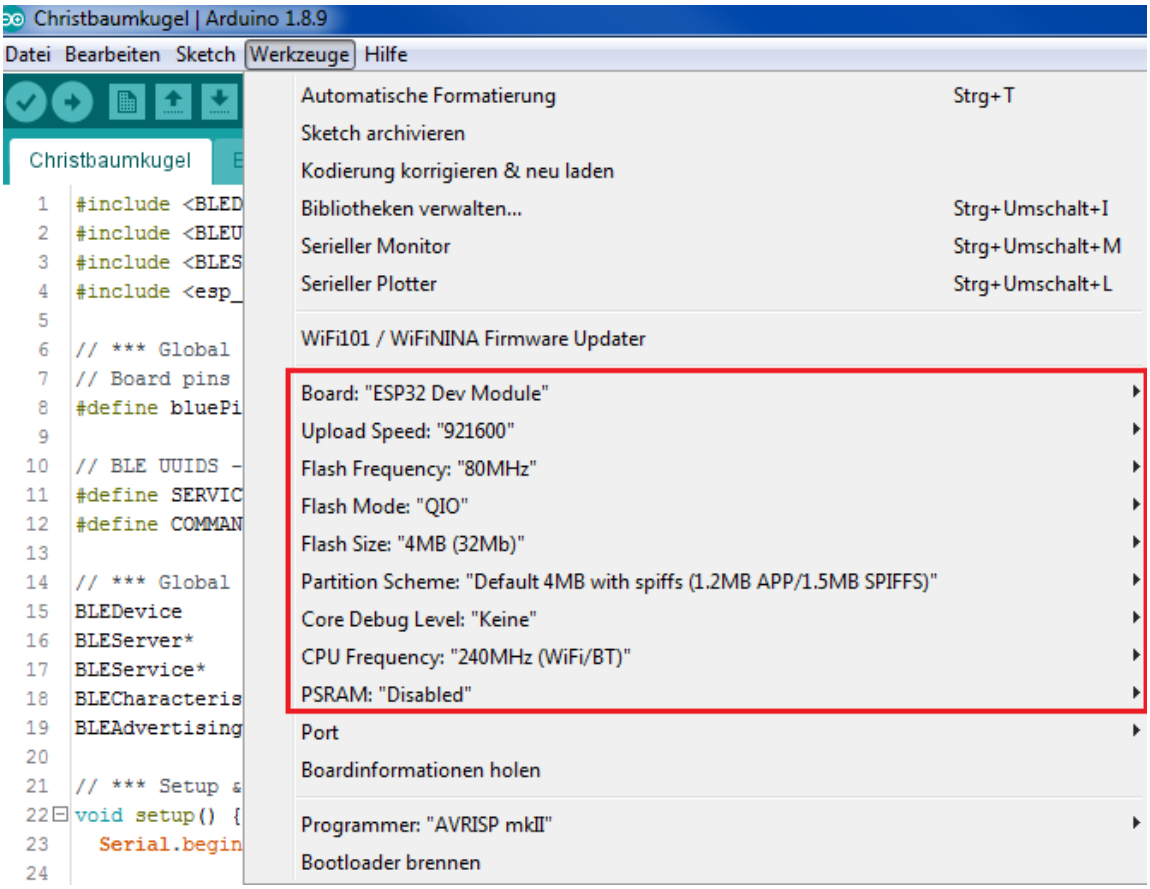


Abbildung 4: Boardkonfiguration

Will man den Sketch auf das Board flashen, klickt man auf den Pfeil „Hochladen“ (oben links in der IDE). Daraufhin wird der Sketch kompiliert und das Flashen beginnt. Will man lediglich kompilieren, klickt man auf den Haken „Überprüfen“.

Während des Flashvorgangs wird am unteren Bildschirm der aktuelle Vorgang in der Konsole ausgegeben. Sobald dort „Connecting...“ steht, muss in einigen Fällen der „Boot“ Knopf des Boards gedrückt werden, bis die Übertragung startet. Danach kann der Knopf losgelassen werden.

Abbildung 5: Konsolenausgabe beim Flashen

Eine praktische Debugmöglichkeit bietet der „Serielle Monitor“, der in der rechten oberen Ecke zu finden ist. Hier werden alle seriellen Ausgaben des Controllers angezeigt, die der Entwickler erstellt hat.

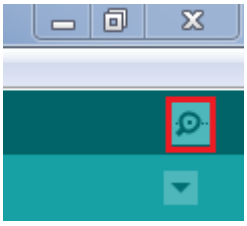


Abbildung 6: Button "Serieller Monitor"

Für Linux Nutzer:

Unter Linux muss zunächst die Versionskontrollsoftware Git und das Python-Modul Serial mittels

```
sudo apt install git python-serial
```

installiert werden.

Damit nun die IDE weiß für welches Board tatsächlich entwickelt wird, gibt man abschließend unter „Werkzeuge“ die korrekte Konfiguration wie in Abbildung 4 dargestellt an. Wichtig hierbei ist zudem das Auswählen des richtigen virtuelle COM Ports, an dem das Board angeschlossen ist.

2.4 Sag Hallo, zur Welt!

Zum Testen der Konfiguration kann ein kleiner „Hello-World“ Sketch implementiert werden. Hierzu muss über „Datei -> Neu“ ein neuer Sketch angelegt werden. Mithilfe des in Abbildung 7 dargestelltem Sketch kann eine Serielle Ausgabe geschrieben werden, welche im Seriellen Monitor verfolgt werden kann. Dabei ist darauf zu achten, dass die Baudrate im Seriellen Monitor mit der im Sketch hinterlegten Baudrate (115200) übereinstimmt.

```
void setup() {
 Serial.begin(115200);
 Serial.println("Hello ESP32 World!");
}

void loop() {
Serial.println("Hello");
delay(500);
}
```

Abbildung 7 Hello World Sketch

2.5 Warum löten, wenn man stecken kann?

Ein Breadboard (Steckplatine) ermöglicht es, schnell und einfach Schaltungen zu testen, ohne das gelötet werden muss.

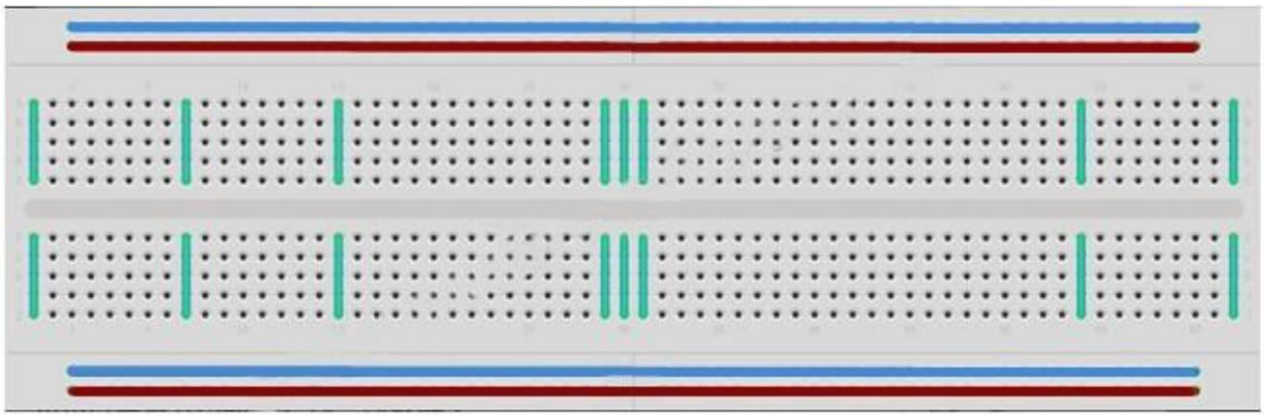


Abbildung 8 Breadboard

Innerhalb des Breadboards befinden sich Metallstreifen, welche die elektrische Verbindung zwischen den Kontaktstellen herstellen. Dabei sind immer die Kontakte innerhalb derselben Reihe (grün eingezeichnet) miteinander verbunden. Wenn du die Pins zweier Bauteile in dieselbe Kontaktreihe steckst, sind diese also miteinander verbunden. Zudem gibt es zwei „Powerlines“ (Rails), welche für die Stromversorgung genutzt werden.

3 PROGRAM YOUR RACER

3.1 Find the right way: Lenkung

Angeschlossen wird der Servo über ein dreipoliges Kabel. Dabei handelt es sich um zwei Versorgungsleitungen - VCC (rot) und GND (braun), sowie eine Datenleitung (gelb).

Mit dem Servo wird eine Winkelbewegung zwischen 0 und 180 Grad gesteuert. Die Neutralposition ist dabei bei 90 Grad erreicht, während 0 und 180 Grad jeweils den Linken und Rechten Anschlag definieren. Bedeutet also, dass wenn du die Servo auf 90 Grad stellst, fährt dein T-Racer geradeaus.

Zur Ansteuerung des Servomotors werden wir die Servo-Klasse aus der Arduino-Servobibliothek " ESP32Servo.h " verwenden.

Beispiel:

```
Servo servo1; //Servo Object
const int servoPin  = <your Pin>;


//attach servo in Setup()
servo1.attach(servoPin, 1000, 2000);


//controll Servo in Loop
 for(int posDegrees = 0; posDegrees <= 180; posDegrees++) {
      servo1.write(posDegrees)
      Serial.println(posDegrees);
      delay(100);
 }
```

3.2 Start your motors: Antrieb

Die digitalen Pins auf dem Entwicklerboard haben eine konstante Ausgangsspannung von 5V bzw. 3.3V. Um die Spannung ändern zu können, unterstützen einige Pins ein PWM-Signal.

Mit dem PWM-Signal und einem MOS-FET können wir die an einen Gleichstrommotor gelieferte Leistung variieren, um so die Geschwindigkeit zu bestimmen. Dadurch könnte die Leistung des Motors angepasst werden, allerdings kann so die Richtung nicht geändert werden. Um die Richtung zu ändern, muss die Polarität getauscht werden, hierzu wird die H-Bridge benötigt.

Motor Variablen definieren:

```
//motor Pins
int motorDir = 18; // → INA
int motorPWM = 5;  // → INB
```

MotorDIR: Richtung des Motors bestimmen (LOW oder HIGH) für Rückwerts oder Vorwärts.

MotorPWM: Drehzahl des Motors bestimmen (Geschwindigkeit) – LOW oder HIGH für volle Geschwindigkeit

Mithilfe der truth table, kann abgeleitet werden, wann der Motor rückwärts bzw. vorwärts fährt.

Table 1 truth table

IA	IB	Output
LO	LO	Off
HI	LO	Forward
LO	HI	Reverse
HI	HI	Off

Vorwärts:

- IA (motorDIR) auf HIGH setzen (5v)
- IB (motorPWM) 255 = Motor aus (siehe truth table), IB (motorPWM) 0 = volle Geschwindigkeit
- Die PWM-Zahl von IB kann nun angepasst werden, um die Leistung des Motors anzupassen → pow

```
digitalWrite( motorDir, HIGH ); // direction = forward
analogWrite( motorPWM, pow );
```

Rückwärts:

- IA (motorDIR) auf LOW setzen (0v)
- IB (motorPWM) 0 = Motor aus (siehe truth table)
- IB = 255 = maximum power → pow

```
digitalWrite( motorDir, LOW ); // direction = backwards
analogWrite( motorPWM, pow );
```

Anhalten:

```
digitalWrite( motorDir, LOW ); // Set motor to off
digitalWrite( motorPWM, LOW );
```

Der Motor kann nun vorwärts und rückwärts betrieben werden indem jeweils die MotorDIR sowie der entsprechende PWM Wert (zwischen 0 und 255) gesetzt wird. Das heißt, wenn wir nur den Wert 0-255 senden, wissen wir nicht, ob der T-Racer vorwärts oder rückwärts fahren muss. Ein weiterer Wert, welcher diese Info enthält, müsste mitgesendet werden. Um dies zu umgehen, sendet die T-Racer App nur einen Wert von 0-255, wobei 128 Stillstand bedeutet. Ist der Wert größer als 128, sollte der T-Racer vorwärtsfahren, ist der Wert kleiner als 128, sollte der T-Racer rückwärtsfahren. Damit trotzdem das volle Spektrum des Motors genutzt werden kann, müssen die Werte entsprechend umgerechnet werden.

```
Vorwärts: pow = 255 - ((fb - 128) * 2);
Rückwärts: pow = 255 - (fb * 2);
```

Achtung: Wie rum du deinen Motor an der H-Bridge anschließt ist wichtig, um später beim Vorwärtsfahren, auch wirklich nach vorne zu fahren. Sollte dein Auto beim vorwärts Befehl, rückwärtsfahren, tausche einfach die Anschlüsse des Motors an der H-Bridge 😉

3.3 Connect to your T-Racer: BLE Verbindung initialisieren

1. **Einbinden aller nötigen Bibliotheken**
   Um die externen Bibliotheken nutzen zu können, müssen diese zu Beginn eingebunden werden.

```
#include <BLEDevice.h>
#include <BLEUtils.h>
#include <BLEServer.h>
```

2. **Variablen definieren**

   Globale Variablen definieren:

```
.....
....
#define SERVICE_UUID        "ae975c80-05f0-11ea-8d71-362b9e155667"
#define COMMAND_CHAR_UUID   "b81f189c-05f0-11ea-8d71-362b9e155667"
BLEDevice         Device;
BLEServer*        pServer;
BLEService*       pService;
BLECharacteristic*  pControllCharacteristic;
BLEAdvertising*     pAdvertising;

setup{
....
....
}

loop{
...
....
}
```

   Achtung: Achte darauf, dass die UUID´s übereinstimmen. Andernfalls kann die App deinen T-Racer nicht finden.

3. **Bluetooth initialisieren**
   Um eine Verbindung mit dem Central einzugehen, muss zunächst das Bluetooth initialisiert werden. Dazu sind mehrere Schritte nötig.

   a. Bluetooth Gerät erstellen

   Zunächst muss das BLE Gerät initialisiert werden. Dabei kann der Name des Gerätes bestimmt werden.

```
Device.init("Name of BLE Device");
```

b. BLE Server erstellen

Nachdem das BLE Gerät erstellt wurde, muss dieses als BLE Server konfiguriert werden. pServer ist dabei vom Typ BLEServer:

```
pServer = Device.createServer();
```

c. Profil definieren

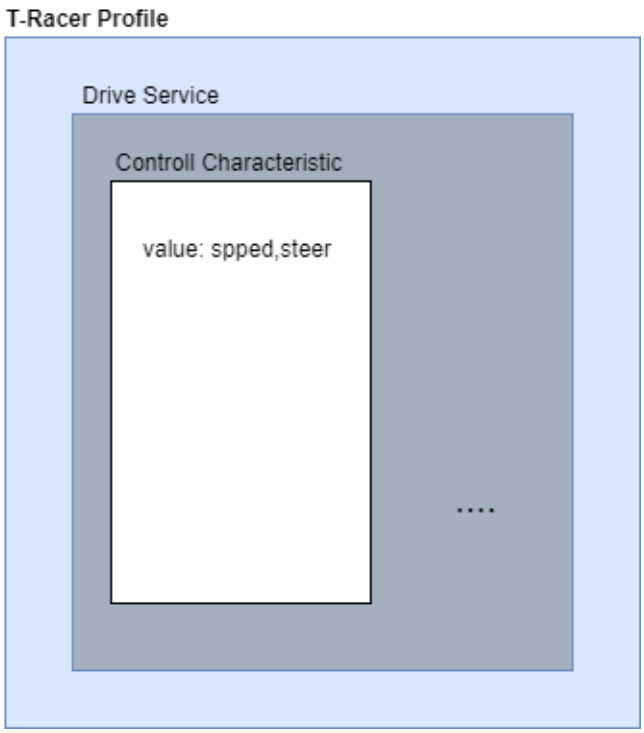


Abbildung 9 T-Racer Profile

```
void initBLE() {
....
.....
pService      = pServer->createService(SERVICE_UUID);
pControllCharacteristic = pService->createCharacteristic(COMMAND_CHAR_UUID,
BLECharacteristic::PROPERTY_WRITE_NR | BLECharacteristic::PROPERTY_WRITE);
// has to be enabled for iOS
pControllCharacteristic->setWriteNoResponseProperty(true);
//Callback Methoden
pServer->setCallbacks(new ServerCallbacks());
pControllCharacteristic ->setCallbacks(new CharacteristicCallbacks());
....
...
}
```

d. UUIDs

Der Service sowie die Charakteristiken benötigen eine UUID, also eine einzigartige ID welche sie repräsentiert. Wie unter Aufgabenpunkt 2 werden dafür Präprozessor Konstanten angelegt. Diese werden direkt unter die oben angelegten Konstanten geschrieben. Damit BLE Client und BLE Server miteinander kommunizieren können, müssen beide das Profil implementieren. Der BLE Client (App) muss daher wissen in welche *Charactersitic* er die entsprechenden Werte schreiben muss. Um dies zu garantieren, müssen folgende UUIDs zugewiesen werden:

Drive ServiceUUID: "ae975c80-05f0-11ea-8d71-362b9e155667"

Controll CharacteristicUUID: "b81f189c-05f0-11ea-8d71-362b9e155667"

e. In den Advertising Zustand wechseln

Ist das Gerät initialisiert und das Profil definiert, kann der Service gestartet und in den Advertising Zustand gewechselt werden, sodass andere BLE Geräte das Gerät durch das Scannen der Umgebung finden können. Die nötigen Befehle können dem Cheat sheet entnommen werden.

Anschließend versendet das Gerät Advertising Nachrichten mit der definierten Nachricht auf den dafür vorgesehen Advertising Kanälen. Durch das Scannen der Umgebung mithilfe der NRFConnect App, sollte das initialisierte Gerät nun sichtbar werden. Dazu muss der Sketch zunächst kompiliert und auf dem Mikrocontroller geflasht werden.

```
void initBLE {

...
...
//start_advertising
Device.startAdvertising();
Serial.println("Start Advertising");

}
```

3.4 Control your T-Racer: Steuerung über BLE

Nachdem du dich erfolgreich mit deinem ESP32 verbinden kannst, wird es Zeit, die übertragenen Werte richtig zu empfangen und zu interpretieren.

1. Werte auf dem BLE-Server empfangen

Um den Wert der Charakteristik beim Empfangen eines neuen Wertes vom Client verarbeiten zu können, muss eine sogenannte Callback Methode definiert werden, welche aufgerufen wird, sobald ein Wert in die Charakteristik geschrieben wird.

Dazu muss zunächst die Callback initialisiert werden:

(**Info**: die Zeile habt ihr schon in der initBLE() Methode 😉)

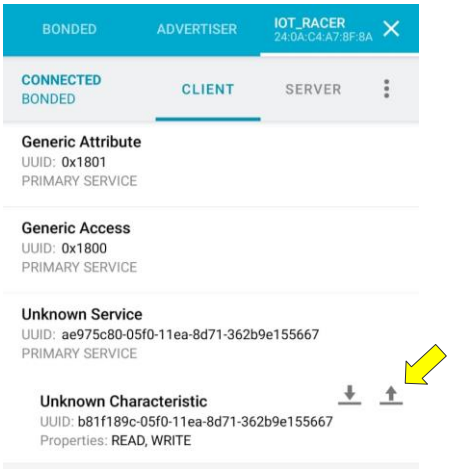
```
pCharCommand->setCallbacks(new CharacteristicCallbacks());
```

Anschließend kann über die Callback-Klasse der Wert ausgelesen werden, indem die Methode onWrite() überlagert wird.

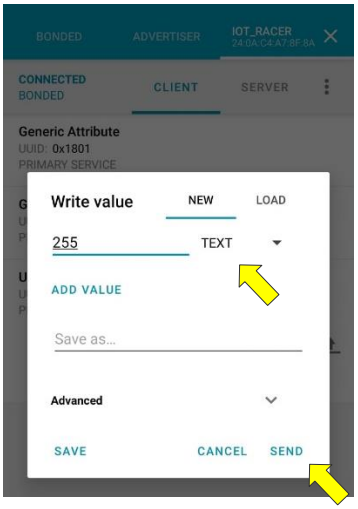
```
class CharacteristicCallbacks: public BLECharacteristicCallbacks {
  void onWrite(BLECharacteristic* pCharacteristic) {
    //receive value via BLE from characteristic
    std::string value = pCharacteristic->getValue();
    //convert to String
    String svalue= value.c_str();
  }
};
```

2. Wert senden, empfangen und ausgeben

Um zu überprüfen, ob die gesendeten Werte richtig empfangen werden, eignet sich die App nRF Connect sehr gut. Damit kannst du dich mit deinem T-Racer verbinden und siehst anschließend das definierte BLE Profil.



Um jetzt einen Wert zu senden, klicke auf den „write“-Pfeil und gebe einen Wert ein. Da serverseitig ein String erwartet wird, wähle UTF8 bzw. Text aus und klicke anschließend auf „send“. Danach solltest du den Wert im seriellen Monitor sehen können.



3. Werte interpretieren

Die T-Racer App sendet die Werte für die Lenkung und den Speed gebündelt in einer Charakteristik im folgenden Format: `<speed,steer>`

Daher müssen die Werte zuerst geparsed und dann an die Servo und den DC-Motor übergeben werden z. B über eine Methode `carControll(speed, steer)`

A CHEATSHEET

Allgemein:

Serial.begin(9600)	Baudrate für die serielle Übertragung definieren.
Serial.println(“STRING“) Serial.print(“STRING“)	Gibt STRING auf dem seriellen Monitor aus. Ohne und mit Zeilenumbruch. Gut für Debugging.

Servo:

#include <ESP32Servo.h>	Include Servo Bibliothek
servo.attach(pin)	Servo dem entsprechenden Pin zuweisen. - *servo*: eine Variable vom Typ Servo - *pin*: Die Pin-Nr. welche der Servo zugewiesen werden soll
servo.write(angle)	Schreibt einen Wert und steuert die Welle entsprechend. Bei einem Standardservo wird dadurch der Winkel der Welle (in Grad) eingestellt und die Welle in diese Ausrichtung bewegt. - Servo: eine Variable vom Typ Servo - Winkel: der auf das Servo zu schreibende Wert, von 0 bis 18

DC Motor:

pinMode(motorDir,OUTPUT) pinMode(motorPWM,OUTPUT)	Configures the specified pin as input or output.
int pow = 255 - ((fb - 128) * 2)	Calculate the value when move forward
int pow = 255 - (fb * 2)	Calculate the value when move backward

Bluetooth Low Energy:

Dokumentation: https://github.com/nkolban/esp32-snippets/tree/master/Documentation

Device.init("T-Racer");
pServer = Device.createServer();
pService = pServer-> createService(SERVICE_UUID);
	Erstellt ein BLE Device namens „T-Racer“, fügt einen Server sowie einen Service mit einer UUID hinzu.
	Typ: BLEDevice

BLECharacteristic* pCharCommand; BLEServer* pServer; pCharCommand = pService-> createCharacteristic(COMMAND_CHAR_UUID, BLECharacteristic::PROPERTY_WRITE | BLECharacteristic::PROPERTY_WRITE_NR | BLECharacteristic::PROPERTY_READ);	Erstellt eine Charakteristik für den Service pService. Parameter: - COMMAND_CHAR_UUID: Die UUID der Charakteristik. - PROPERTY_WRITE: Die Charakteristik hat Schreibzugriff. - PROPERTY_WRITE_NR: Schreibzugriff ohne Response
BLEService* pService; pService->start();	Startet den Service.

Device.init("T-Racer");
pServer = Device.createServer();
pService = pServer-> createService(SERVICE_UUID);
	Erstellt ein BLE Device namens „T-Racer“, fügt einen Server sowie einen Service mit einer UUID hinzu.
	Typ: BLEDevice

BLEAdvertising* pAdvertising;
 	
pAdvertising = Device.getAdvertising();
pAdvertising-> addServiceUUID(SERVICE_UUID);
pAdvertising->setScanResponse(true);
pAdvertising->setMinPreferred(0x06);
pAdvertising->setMinPreferred(0x12);
Device.startAdvertising();
	Konfiguriert und startet das BLE Advertising. Notwendig, um von anderen Geräten gesehen zu werden.

pServer->setCallbacks(new ServerCallbacks()); pCharCommand->setCallbacks(new CharacteristicCallbacks());	Callbacks für den Server und die Charakteristik registrieren.

class ServerCallbacks : public BLEServerCallbacks {
  void onConnect(BLEServer* pServerCallback) {
  Serial.println("Client connected");
  }
};
	Beispiel Server-Callback, der bei Verbindung eines Clients eine Meldung auf dem seriellen Monitor ausgibt.

class CharacteristicCallbacks : public BLECharacteristicCallbacks
{
  void onWrite(BLECharacteristic* pCharacteristic){
  std::string value = pCharacteristic->getValue();
  }
};
	Beispielhafter Charakteristik Callback, welcher ein onWirte-event abfängt und den Wert ausliest.