

# Mit MQTT einen Roboter steuern [Teil 3]

Im ersten und zweiten Teil dieser Blogserie wurde Ihnen das Grundwissen über MQTT und wie es mit verschiedenen MicroControllern genutzt werden kann erklärt. In diesem Blogbeitrag wird der erwähnte (Roll-)Roboter das Licht der Welt erblicken und mit einer simplen Fernbedienung gesteuert werden.

Damit dieses Vorhaben auch gelingt, ist ein MQTT-Broker notwendig. Wie Sie diesen ganz einfach mit einem Raspberry Pi umsetzen, ist in Teil 1 dieser Blogserie genaustens erklärt.

## Hardware-Voraussetzung

Für diesen Blogbeitrag brauchen Sie die folgenden Bauteile, siehe Tabelle 1.

Pos	Anzahl	Bauteil	Link
1	1	Raspberry Pi (erforderlich)	<a href="https://www.az-delivery.de/">https://www.az-delivery.de/</a>
2	1	Passendes Netzteil	
3	2	ESP32 NodeMCU Module WLAN WiFi Development Board (erforderlich)	<a href="https://www.az-delivery.de/">https://www.az-delivery.de/</a>
4	2	Potentiometer (erforderlich)	<a href="https://www.az-delivery.de/">https://www.az-delivery.de/</a>
5	1	Eine H-Brücke L298N	<a href="https://www.amazon.de/">https://www.amazon.de/</a>
6	1	Ein Breadboard	<a href="https://www.az-delivery.de/">https://www.az-delivery.de/</a>
7	1	Einen Rollroboterbausatz	<a href="https://www.amazon.de/">https://www.amazon.de/</a>
8	1	Einige Jumper wires Female to Female	<a href="https://www.az-delivery.de/">https://www.az-delivery.de/</a>
9	1	Powerbank fürs Handy	

*Tabelle 1: Benötigte Hardware*

Denken Sie bei dem Raspberry Pi daran, dass Sie neben der oben genannten Hardware auch eine MicroSD-Karte und Spannungsversorgung brauchen. Hierzu sollten Sie das Raspberry Pi OS (ehemals Raspbian) als Image auf die Karte kopieren und entsprechend einen MQTT-Broker nach Anleitung aus Teil 1 dieser Serie installieren.

## Software-Voraussetzung

Für den Blogbeitrag benötigen Sie folgende Software:

- Arduino IDE (<https://www.arduino.cc/en/Main/Software>), hier am besten die aktuellste Version herunterladen
- Die Bibliothek **PubSubClient** mit allen Abhängigkeiten

Wie Sie Libraries über die Bibliotheksverwaltung installieren können, ist unter <https://www.az-delivery.de/blogs/azdelivery-blog-fur-arduino-und-raspberry-pi/arduino-ide-programmieren-fuer-einsteiger-teil-1> Abschnitt Bibliotheksverwaltung, näher beschrieben.

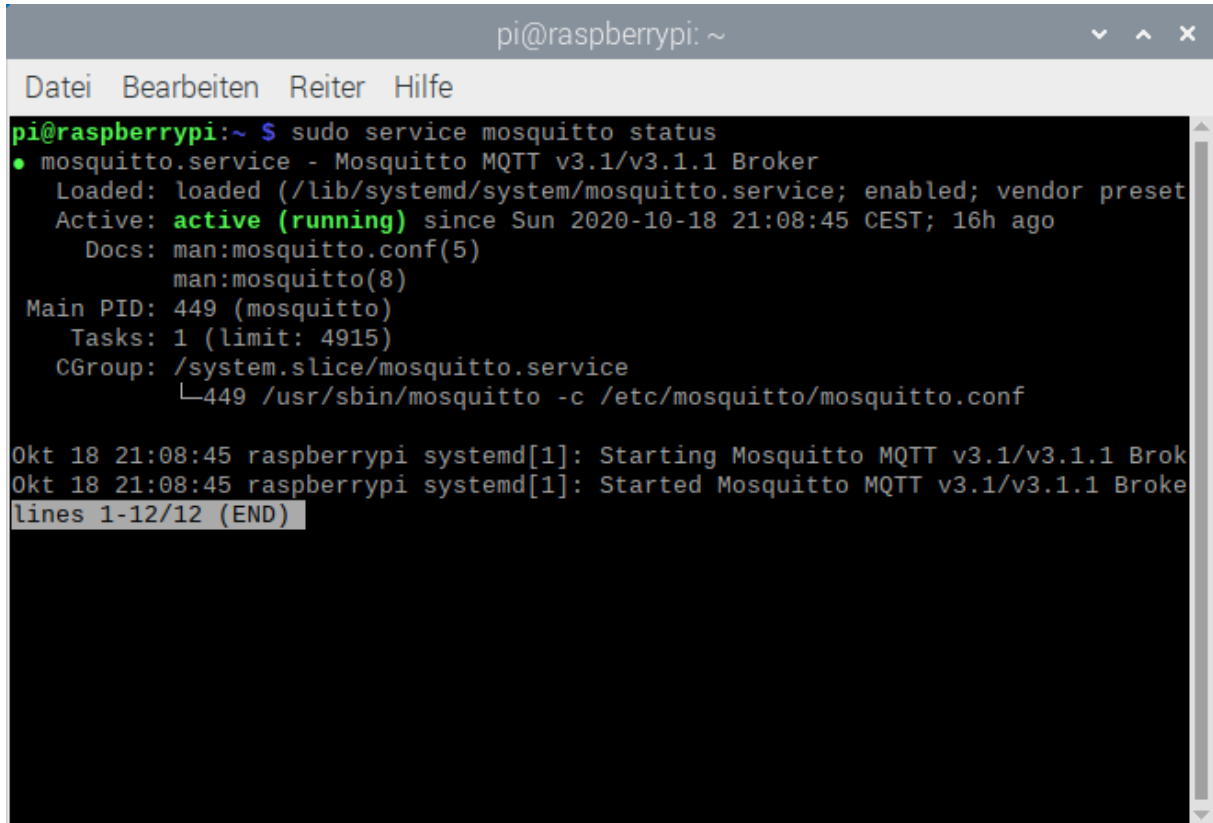
## Grundvoraussetzung

Prüfen Sie bitte zunächst, ob der MQTT-Broker beim Hochfahren vom Raspberry Pi gestartet wurde. Geben Sie dazu den Befehl aus Code 1 ins Terminal ein.

```
sudo service mosquitto status
```

Code 1: Abfrage im Terminal, ob mosquitto gestartet ist

Sollte die Ausgabe ein **active (running)** zeigen, siehe Abbildung 1, sind keine weiteren Kommandos nötig.

The image shows a terminal window titled 'pi@raspberrypi: ~'. The terminal output of the command 'sudo service mosquitto status' is as follows:

```
pi@raspberrypi:~ $ sudo service mosquitto status
• mosquitto.service - Mosquitto MQTT v3.1/v3.1.1 Broker
   Loaded: loaded (/lib/systemd/system/mosquitto.service; enabled; vendor preset:
   Active: active (running) since Sun 2020-10-18 21:08:45 CEST; 16h ago
     Docs: man:mosquitto.conf(5)
           man:mosquitto(8)
   Main PID: 449 (mosquitto)
    Tasks: 1 (limit: 4915)
   CGroup: /system.slice/mosquitto.service
           └─449 /usr/sbin/mosquitto -c /etc/mosquitto/mosquitto.conf

Okt 18 21:08:45 raspberrypi systemd[1]: Starting Mosquitto MQTT v3.1/v3.1.1 Brok
Okt 18 21:08:45 raspberrypi systemd[1]: Started Mosquitto MQTT v3.1/v3.1.1 Broke
lines 1-12/12 (END)
```

Abbildung 1: Status von mosquitto-Broker im Terminal

Sollte sich die Ausgabe unterscheiden, so prüfen Sie bitte noch einmal, ob mosquitto korrekt installiert wurde und auch der Dienst korrekt in den Autostart eingebunden wurde. Die genaue Anleitung finden Sie in Teil 1.

## Die Fernbedienung

Damit unser Rollrobooter später auch gesteuert werden kann, braucht es eine Fernbedienung. Diese muss zwei Aufgaben erfüllen:

1. Die Geschwindigkeit für Vorwärts, Rückwärts, links und rechts erfassen
2. Die Daten an den MQTT-Broker senden

In diesem Fall wird die Bewegungsrichtung über zwei Potentiometer realisiert, siehe Abbildung 2.

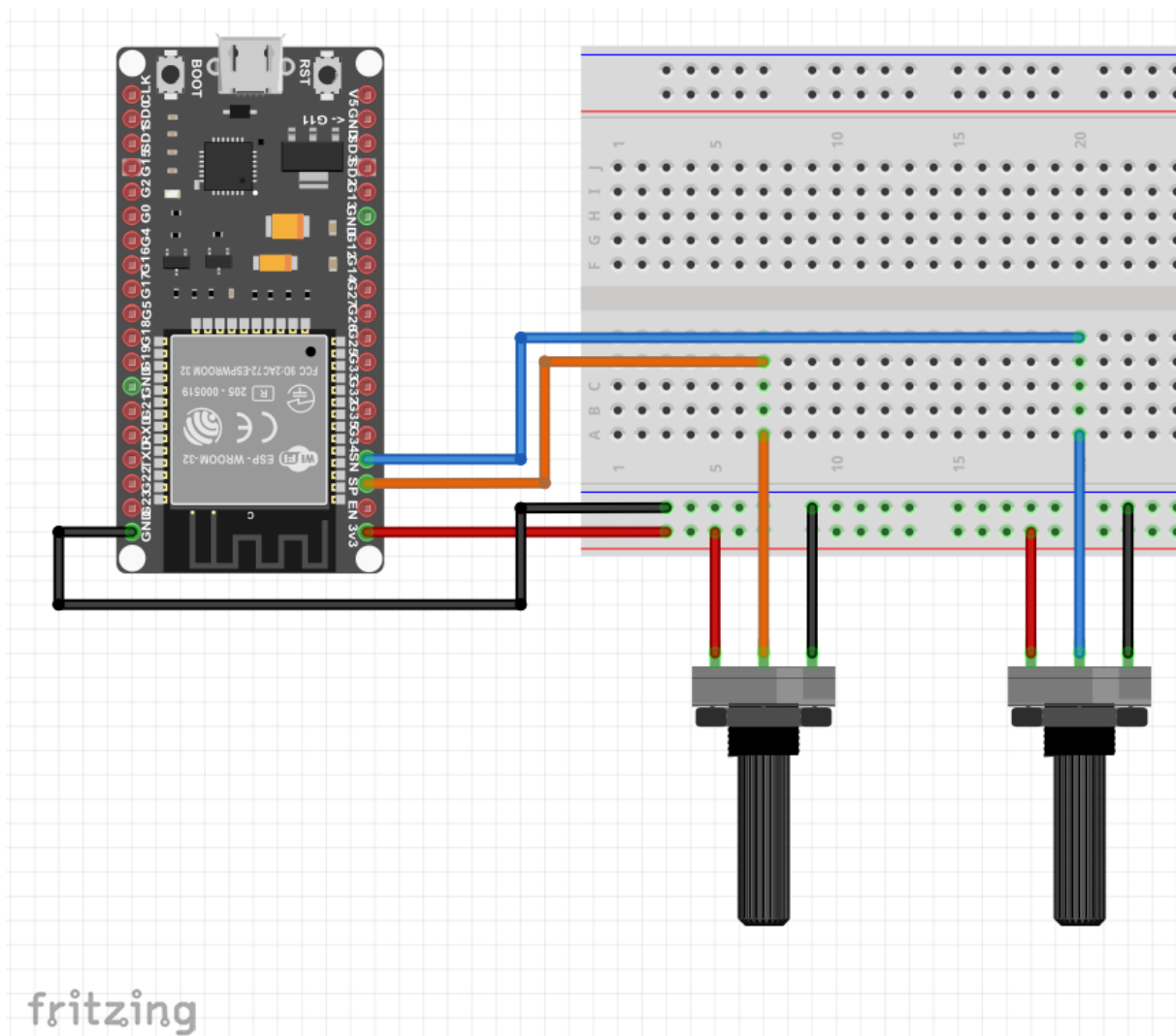


Abbildung 2: Verdrahtung Fernbedienung

Als MicroController kommt ein ESP32 NodeMCU Module WLAN WiFi Development Board zum Einsatz. Denn dieser MicroController verfügt über mehr als einen analogen Eingang und hat das WLAN direkt auf dem Controller verbaut. Dadurch spart man sich, im Vergleich zu anderen MicroControllern, das zusätzliche Verkabeln eines WiFi-Boards.

Code 2 zeigt den kompletten Quellcode der Fernbedienung. Damit der Code auch bei Ihnen funktioniert, müssen Sie in den Zeilen mit den Kommentaren „Enter Wifi-Name“, „Enter Passkey“ und „Name of the mqtt broker“ ihre „WiFi-Credentials“ und MQTT-Broker-Namen zwischen den Anführungszeichen „“ einfügen.

```
//-----
// ESP-NodeMCU remote controller
// mqtt-broker and mapping analog input
// Autor: Joern Weise
// License: GNU GPI 3.0
// Created: 20. Jan 2021
// Update: 20. Jan 2021
//-----
#include <WiFi.h>
#include <PubSubClient.h> //Lib for MQTT Pub and Sub
```

```

//Define WiFi-Settings
#ifndef STASSID
#define STASSID "" //Enter Wfi-Name
#define STAPSK "" //Enter Passkey
#endif

#define ADVANCEDIAG 1

#define ADC_STRAIGHT 36
#define ADC_CROSS 39

const char* MQTT_BROKER = ""; //Name of the mqtt broker
const char* PubTopicStraight = "/RemoteControl/Straight"; //Topic first temp
const char* PubTopicCross = "/RemoteControl/Cross"; //Topic second temp
String clientID = "RemoteController"; //Clientname for MQTT-Broker

int iLastStraight, iLastCross;
//Create objects for mqtt
WiFiClient espClient;
PubSubClient mqttClient(espClient);

#define MSG_BUFFER_SIZE (50)
char msg[MSG_BUFFER_SIZE];

void setup()
{
  Serial.begin(115200);
  Serial.println("Remote control started");
  writeAdvanceDiag("Init WiFi", true);
  setupWifi();
  writeAdvanceDiag("Init Wifi - DONE", true);
  writeAdvanceDiag("Set MQTT-Server", true);
  mqttClient.setServer(MQTT_BROKER,1883);
  iLastStraight = -7;
  iLastCross = -7;
  writeAdvanceDiag("Finish setup()-Function", true);
}

void loop() {
  if(!mqttClient.connected())
    reconnectMQTT();

  mqttClient.loop();
  //Read value from analog input and map value
  int iMappedStraight = map(analogRead(ADC_STRAIGHT),4095,0,-2,2);
  if(iMappedStraight != iLastStraight)
  {

```

```

    snprintf(msg,MSG_BUFFER_SIZE, "%1d",iMappedStraight); //Convert message to char
    mqttClient.publish(PubTopicStraight,msg,true); //Send to broker
    writeAdvanceDiag("Send Straight: " + String(iMappedStraight), true);
    iLastStraight = iMappedStraight;
}
//Read value from analog input and map value
int iMappedCross = map(analogRead(ADC_CROSS),4095,0,-2,2);
if(iMappedCross != iLastCross)
{
    snprintf(msg,MSG_BUFFER_SIZE, "%1d",iMappedCross); //Convert message to char
    mqttClient.publish(PubTopicCross,msg,true); //Send to broker
    writeAdvanceDiag("Send Cross: " + String(iMappedCross), true);
    iLastCross = iMappedCross;
}
}

/*
* =====
* Function:   setupWifi
* Returns:   void
* Description: Setup wifi to connect to network
* =====
*/
void setupWifi()
{
    Serial.println("Connection to: " + String(STASSID));
    WiFi.mode(WIFI_STA);
    WiFi.begin(STASSID, STAPSK);
    while (WiFi.status() != WL_CONNECTED)
    {
        delay(500);
        Serial.print(".");
    }
    Serial.println("");
    Serial.println("WiFi connected");
    Serial.println("IP address: ");
    Serial.println(WiFi.localIP());
}

/*
* =====
* Function:   reconnectMQTT
* Returns:   void
* Description: If there is no connection to MQTT, this function is
*              called. In addition, the desired topic is registered.
* =====
*/
void reconnectMQTT()
{

```

```

while(!mqttClient.connected())
{
  writeAdvanceDiag("Login to MQTT-Broker", true);
  if(mqttClient.connect(clientID.c_str()))
  {
    Serial.println("Connected to MQTT-Broker " + String(MQTT_BROKER));
  }
  else
  {
    writeAdvanceDiag("Failed with rc=" +String(mqttClient.state()), true);
    Serial.println("Next MQTT-Connect in 3 sec");
    delay(3000);
  }
}
}

/*
* =====
* Function:   writeAdvanceDiag
* Returns:    void
* Description: Writes advance msg to serial monitor, if
*             ADVANCEDIAG >= 1
* msg:        Message for the serial monitor
* newLine:     Message with linebreak (true)
* =====
*/
void writeAdvanceDiag(String msg, bool newLine)
{
  if(bool(ADVANCEDIAG)) //Check if advance diag is enabled
  {
    if(newLine)
      Serial.println(msg);
    else
      Serial.print(msg);
  }
}

```

*Code 2: Quellcode Fernbedienung*

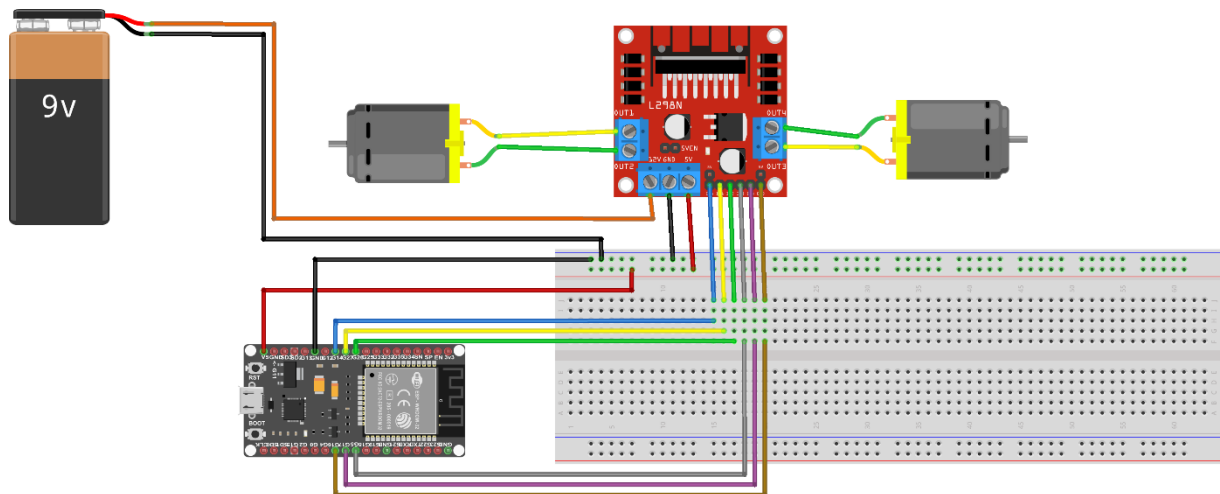
Die Kernfunktion versteckt sich in der Funktion loop(). Hier wird zyklisch der aktuelle Wert der Potentiometer auf den beiden analogen Inputs eingelesen und mit dem letzten bekannten Werten verglichen. Wenn es eine Änderung gibt, wird dieser Wert gemappt und an den MQTT-Broker gesendet. Vom Grundprinzip ist es dasselbe Verhalten wie das Potentiometer aus Teil 2, nur dass vorher noch der analoge Wert gemappt und erst dann an den Broker übermittelt wird. Damit später die Fernbedienung überall benutzen werden kann, wird diese mit einer Powerbank betrieben. Diese liefert 5V an einem USB-Ausgang und versorgt so den MicroController mit Strom.

## Der Rollrobooter

Die Umsetzung des Rollroboters soll ebenso einfach sein wie die Fernbedienung. Vom Prinzip müssen zwei Aufgaben vom MicroController des Rollroboters übernommen werden:

1. Empfangen der Werte der Fernbedienung über den MQTT-Broker
2. Die empfangenen Werte so umsetzen, dass die Motoren korrekt drehen

Damit Punkt 2 erfüllt werden kann, wird hier eine H-Brücke L298N verwendet. Was eine H-Brücke genau ist, wird in einem späteren Blog noch einmal genau erklärt; vorerst reicht es zu wissen, dass dieses elektrische Bauteil für die Ansteuerung der Motoren zuständig ist. Gleichzeitig erspart die H-Brücke eine zweite Spannungsquelle, da bei einer Spannung bis zu 12V die H-Brücke L298N 5V an einen separaten Ausgang liefert. Theoretisch kann man aber auch für MicroController und H-Brücke zwei getrennte Spannungsquellen verwenden. Wichtig dabei ist, dass beide auf das gleiche Potenzial gehoben werden, dazu müssen beide GND-Anschlüsse zusammengeschaltet werden. Abbildung 3 zeigt die Verdrahtung vom Rollroboter mit nur einer Spannungsquelle.



fritzing

Abbildung 3: Verdrahtung Rollroboter

Achten Sie bitte darauf, dass die 5V-Spannung an den angegebenen Pins vom ESP32 NodeMCU Module WLAN WiFi Development Board angeschlossen sind. Nutzen Sie nicht den GND-Pin direkt neben V5, sonst wird der MicroController nicht booten. Bei der H-Brücke L298N muss zusätzlich der Jumper für 5V gesteckt sein, ansonsten erhalten Sie keine 5V-Spannung vom Modul, siehe Abbildung 4 rote Umrandung. Bei Spannungen über 5V müssen Sie zwingend den Jumper entfernen, da sonst Bauteile auf dem Modul beschädigt werden.

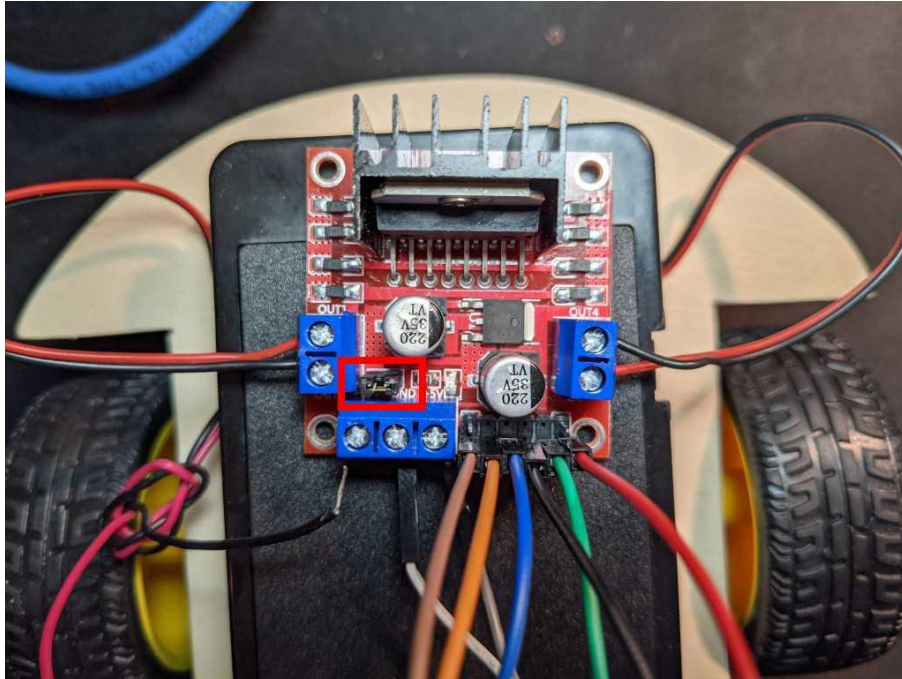


Abbildung 4: 5V-Jumper auf L298N

```

Den Quellcode für den Rollroboter sehen Sie in //-----
// ESP-NodeMCU robot
// mqtt-broker and mapping analog input
// Autor: Joern Weise
// License: GNU GPI 3.0
// Created: 20. Jan 2021
// Update: 29. Jan 2021
//-----
#include <WiFi.h>
#include <PubSubClient.h> //Lib for MQTT Pub and Sub

//Define WiFi-Settings
#ifndef STASSID
#define STASSID "" //Enter Wifi-Name
#define STAPSK "" //Enter Passkey
#endif

#define ADVANCEDIAG 1

#define MAXSPEED 255
#define MINSPEED 155
//MQTT stuff
const char* MQTT_BROKER = ""; //Name of the mqtt broker
String clientID = "AZBot"; //Clientname for MQTT-Broker
const char* SubTopicStraight = "/RemoteControl/Straight"; //Topic first temp
const char* SubTopicCross = "/RemoteControl/Cross"; //Topic second temp

int iMQTTStraight, iMQTTCross, iMQTTStraightNew, iMQTTCrossNew, iMQTTStraightLast,
iMQTTCrossLast;

//Create objects for mqtt

```



```

WiFiClient espClient;
PubSubClient mqttClient(espClient);

//Timer-vars for debounce
unsigned long ulDebounce = 10; //Debounce-time
unsigned long ulLastDebounceTimeStraight, ulLastDebounceTimeCross; //Timer to debouce
button

//PWM and motor configuration
// Motor A
const int motor1Pin1 = 27;
const int motor1Pin2 = 26;
const int enable1Pin = 14;
const int motor1channel = 0;
// Motor B
const int motor2Pin1 = 17;
const int motor2Pin2 = 5;
const int enable2Pin = 16;
const int motor2channel = 1;

// Setting PWM properties
const int freq = 30000;
const int resolution = 8;

bool bUpdateMovement = false; //Will set, if there are new movements from mqtt available
/*
=====
Function:  setup
Returns:  void
Description: Needed setup-function
=====
*/
void setup()
{
    // sets the pins as outputs:
    pinMode(motor1Pin1, OUTPUT);
    pinMode(motor1Pin2, OUTPUT);
    pinMode(enable1Pin, OUTPUT);
    pinMode(motor2Pin1, OUTPUT);
    pinMode(motor2Pin2, OUTPUT);
    pinMode(enable2Pin, OUTPUT);
    Serial.begin(115200);
    Serial.println("Remote control started");
    iMQTTStraightNew = 0;
    iMQTTCrossNew = 0;
    writeAdvanceDiag("Init WiFi", true);
    setupWifi();
    writeAdvanceDiag("Init Wifi - DONE", true);
    writeAdvanceDiag("Set MQTT-Server", true);
    mqttClient.setServer(MQTT_BROKER, 1883);
    writeAdvanceDiag("Set Callback-function", true);
    mqttClient.setCallback(callback);
    writeAdvanceDiag("Set PWM-Channels", true);

```

```

ledcSetup(motor1channel, freq, resolution); //Configure PWM for motor 1
ledcSetup(motor2channel, freq, resolution); //Configure PWM for motor 2
ledcAttachPin(enable1Pin, motor1channel); //Attach channel 1 to motor 1
ledcAttachPin(enable2Pin, motor2channel); //Attach channel 2 to motor 2

writeAdvanceDiag("Finish setup()-Function", true);
}

/*
=====
Function:   loop
Returns:   void
Description: Needed loop-function
=====
*/
void loop()
{
  if (!mqttClient.connected())
    reconnectMQTT();

  mqttClient.loop();
  DebounceStraight();
  DebounceCross();
  int iSpeedMotor1, iSpeedMotor2;
  if (bUpdateMovement) //Check if there is a new movement available from mqtt
  {
    Serial.println("Current value straight: " + String(iMQTTStraight));
    Serial.println("Current value cross: " + String(iMQTTCross));
    if (iMQTTStraight != 0)
    {
      if (iMQTTStraight < 0)
      {
        digitalWrite(motor1Pin1, LOW);
        digitalWrite(motor1Pin2, HIGH);
        digitalWrite(motor2Pin1, LOW);
        digitalWrite(motor2Pin2, HIGH);
      }
      else
      {
        digitalWrite(motor1Pin1, HIGH);
        digitalWrite(motor1Pin2, LOW);
        digitalWrite(motor2Pin1, HIGH);
        digitalWrite(motor2Pin2, LOW);
      }
    }
    if (abs(iMQTTStraight) == 1)
    {
      iSpeedMotor1 = MAXSPEED - (MAXSPEED - MINSPEED)/2;
      iSpeedMotor2 = MAXSPEED - (MAXSPEED - MINSPEED)/2;
    }
    else
    {
      iSpeedMotor1 = MAXSPEED;
      iSpeedMotor2 = MAXSPEED;
    }
  }
}

```

```

    }
}
else
{
    iSpeedMotor1 = 0;
    iSpeedMotor2 = 0;
}

if (iMQTTCross != 0)
{
    if (iMQTTCross < 0)
    {
        if(iSpeedMotor1 == MAXSPEED)
        {
            if(abs(iMQTTCross) == 1)
                iSpeedMotor1 = MAXSPEED - (MAXSPEED - MINSPEED)/2;
            else
                iSpeedMotor1 = MINSPEED;
        }
        else
        {
            if(abs(iMQTTCross) == 1)
                iSpeedMotor1 = MINSPEED;
            else
                iSpeedMotor1 = 0;
        }
        Serial.println("New Speed motor 1: " + String(iSpeedMotor1));
    }
    else
    {
        if(iSpeedMotor2 == MAXSPEED)
        {
            if(abs(iMQTTCross) == 1)
                iSpeedMotor2 = MAXSPEED - (MAXSPEED - MINSPEED)/2;
            else
                iSpeedMotor2 = MINSPEED;
        }
        else
        {
            if(abs(iMQTTCross) == 1)
                iSpeedMotor2 = MINSPEED;
            else
                iSpeedMotor2 = 0;
        }
        Serial.println("New Speed motor 2: " + String(iSpeedMotor2));
    }
}
//Write speed to motor pwm
ledcWrite(motor1channel, iSpeedMotor1);
ledcWrite(motor2channel, iSpeedMotor2);
bUpdateMovement = false; //New movement set
}

```

```

}

/*
=====
Function:  setupWifi
Returns:  void
Description: Setup wifi to connect to network
=====
*/
void setupWifi()
{
  Serial.println("Connection to: " + String(STASSID));
  WiFi.mode(WIFI_STA);
  WiFi.begin(STASSID, STAPSK);
  while (WiFi.status() != WL_CONNECTED)
  {
    delay(500);
    Serial.print(".");
  }
  Serial.println("");
  Serial.println("WiFi connected");
  Serial.println("IP address: ");
  Serial.println(WiFi.localIP());
}

/*
=====
Function:  callback
Returns:  void
Description: Will automatical called, if a subscribed topic
             has a new message
topic:     Returns the topic, from where a new msg comes from
payload:   The message from the topic
length:    Length of the msg, important to get content
=====
*/
void callback(char* topic, byte* payload, unsigned int length)
{
  String strMessage = "";
  writeAdvanceDiag("Message arrived from topic: " + String(topic), true);
  writeAdvanceDiag("Message length: " + String(length), true);
  for (int i = 0; i < length; i++)
    strMessage += String((char)payload[i]);
  writeAdvanceDiag("Message is: " + strMessage, true);
  if (String(topic) == String(SubTopicStraight))
  {
    iMQTTStraightNew = strMessage.toInt();
  }
  else if (String(topic) == String(SubTopicCross))
  {
    iMQTTCrossNew = strMessage.toInt();
  }
}

```

```

/*
=====
Function:  reconnectMQTT
Returns:   void
Description: If there is no connection to MQTT, this function is
            called. In addition, the desired topic is registered.
=====
*/
void reconnectMQTT()
{
    while (!mqttClient.connected())
    {
        writeAdvanceDiag("Login to MQTT-Broker", true);
        if (mqttClient.connect(clientID.c_str()))
        {
            Serial.println("Connected to MQTT-Broker " + String(MQTT_BROKER));
            writeAdvanceDiag("Subscribe topic '" + String(SubTopicStraight) + "'", true);
            mqttClient.subscribe(SubTopicStraight, 1); //Subscribe topic "SubTopicStraight"
            writeAdvanceDiag("Subscribe topic '" + String(SubTopicCross) + "'", true);
            mqttClient.subscribe(SubTopicCross, 1); //Subscribe topic "SubTopicCross"
        }
        else
        {
            writeAdvanceDiag("Failed with rc=" + String(mqttClient.state()), true);
            Serial.println("Next MQTT-Connect in 3 sec");
            delay(3000);
        }
    }
}

/*
=====
Function:  writeAdvanceDiag
Returns:   void
Description: Writes advance msg to serial monitor, if
            ADVANCEDIAG >= 1
msg:       Message for the serial monitor
newLine:   Message with linebreak (true)
=====
*/
void writeAdvanceDiag(String msg, bool newLine)
{
    if (bool(ADVANCEDIAG)) //Check if advance diag is enabled
    {
        if (newLine)
            Serial.println(msg);
        else
            Serial.print(msg);
    }
}

```

```

/*
=====
Function:  DebounceStraight
Returns:   void
Description: Set new value, if debounce is over
            If there is a new valid bUpdateMovement
            will set true
=====
*/
void DebounceStraight()
{
    if (iMQTTStraightNew != iMQTTStraightLast)
        ulLastDebounceTimeStraight = millis();

    if ((millis() - ulLastDebounceTimeStraight) > ulDebounce)
    {

        if (iMQTTStraightNew != iMQTTStraight)
        {
            iMQTTStraight = iMQTTStraightNew;
            writeAdvanceDiag("New straight value " + String(iMQTTStraight), true);
            bUpdateMovement = true;
        }
    }
    iMQTTStraightLast = iMQTTStraightNew;
}

/*
=====
Function:  DebounceCross
Returns:   void
Description: Set new value, if debounce is over
            If there is a new valid bUpdateMovement
            will set true
=====
*/
void DebounceCross()
{
    if (iMQTTCrossNew != iMQTTCrossLast)
        ulLastDebounceTimeCross = millis();

    if ((millis() - ulLastDebounceTimeCross) > ulDebounce)
    {
        if (iMQTTCrossNew != iMQTTCross)
        {
            iMQTTCross = iMQTTCrossNew;
            writeAdvanceDiag("New cross value " + String(iMQTTCross), true);
            bUpdateMovement = true;
        }
    }
    iMQTTCrossLast = iMQTTCrossNew;
}

```

Code 3. Geben Sie auch hier die Einstellungen von Ihrem Netzwerk in den Zeilen mit den Kommentaren „Enter Wifi-Name“, „Enter Passkey“ und „Name of the mqtt broker“.

```
//-----  
// ESP-NodeMCU robot  
// mqtt-broker and mapping analog input  
// Autor: Joern Weise  
// License: GNU GPI 3.0  
// Created: 20. Jan 2021  
// Update: 29. Jan 2021  
//-----  
#include <WiFi.h>  
#include <PubSubClient.h> //Lib for MQTT Pub and Sub  
  
//Define WiFi-Settings  
#ifndef STASSID  
#define STASSID "" //Enter Wifi-Name  
#define STAPSK "" //Enter Passkey  
#endif  
  
#define ADVANCEDIAG 1  
  
#define MAXSPEED 255  
#define MINSPEED 155  
//MQTT stuff  
const char* MQTT_BROKER = ""; //Name of the mqtt broker  
String clientID = "AZBot"; //Clientname for MQTT-Broker  
const char* SubTopicStraight = "/RemoteControl/Straight"; //Topic first temp  
const char* SubTopicCross = "/RemoteControl/Cross"; //Topic second temp  
  
int iMQTTStraight, iMQTTCross, iMQTTStraightNew, iMQTTCrossNew, iMQTTStraightLast,  
iMQTTCrossLast;  
  
//Create objects for mqtt  
WiFiClient espClient;  
PubSubClient mqttClient(espClient);  
  
//Timer-vars for debounce  
unsigned long ulDebounce = 10; //Debounce-time  
unsigned long ulLastDebounceTimeStraight, ulLastDebounceTimeCross; //Timer to debouce  
button  
  
//PWM and motor configuration  
// Motor A  
const int motor1Pin1 = 27;  
const int motor1Pin2 = 26;  
const int enable1Pin = 14;  
const int motor1channel = 0;  
// Motor B  
const int motor2Pin1 = 17;  
const int motor2Pin2 = 5;  
const int enable2Pin = 16;
```

```

const int motor2channel = 1;

// Setting PWM properties
const int freq = 30000;
const int resolution = 8;

bool bUpdateMovement = false; //Will set, if there are new movements from mqtt available
/*
=====
Function:  setup
Returns:   void
Description: Needed setup-function
=====
*/
void setup()
{
    // sets the pins as outputs:
    pinMode(motor1Pin1, OUTPUT);
    pinMode(motor1Pin2, OUTPUT);
    pinMode(enable1Pin, OUTPUT);
    pinMode(motor2Pin1, OUTPUT);
    pinMode(motor2Pin2, OUTPUT);
    pinMode(enable2Pin, OUTPUT);
    Serial.begin(115200);
    Serial.println("Remote control started");
    iMQTTStraightNew = 0;
    iMQTTCrossNew = 0;
    writeAdvanceDiag("Init WiFi", true);
    setupWifi();
    writeAdvanceDiag("Init Wifi - DONE", true);
    writeAdvanceDiag("Set MQTT-Server", true);
    mqttClient.setServer(MQTT_BROKER, 1883);
    writeAdvanceDiag("Set Callback-function", true);
    mqttClient.setCallback(callback);
    writeAdvanceDiag("Set PWM-Channels", true);
    ledcSetup(motor1channel, freq, resolution); //Configure PWM for motor 1
    ledcSetup(motor2channel, freq, resolution); //Configure PWM for motor 2
    ledcAttachPin(enable1Pin, motor1channel); //Attach channel 1 to motor 1
    ledcAttachPin(enable2Pin, motor2channel); //Attach channel 2 to motor 2

    writeAdvanceDiag("Finish setup()-Function", true);
}

/*
=====
Function:  loop
Returns:   void
Description: Needed loop-function
=====
*/
void loop()
{
    if (!mqttClient.connected())

```



```

reconnectMQTT();

mqttClient.loop();
DebounceStraight();
DebounceCross();
int iSpeedMotor1, iSpeedMotor2;
if (bUpdateMovement) //Check if there is a new movement available from mqtt
{
    Serial.println("Current value straight: " + String(iMQTTStraight));
    Serial.println("Current value cross: " + String(iMQTTCross));
    if (iMQTTStraight != 0)
    {
        if (iMQTTStraight < 0)
        {
            digitalWrite(motor1Pin1, LOW);
            digitalWrite(motor1Pin2, HIGH);
            digitalWrite(motor2Pin1, LOW);
            digitalWrite(motor2Pin2, HIGH);
        }
        else
        {
            digitalWrite(motor1Pin1, HIGH);
            digitalWrite(motor1Pin2, LOW);
            digitalWrite(motor2Pin1, HIGH);
            digitalWrite(motor2Pin2, LOW);
        }
        if(abs(iMQTTStraight) == 1)
        {
            iSpeedMotor1 = MAXSPEED - (MAXSPEED - MINSPEED)/2;
            iSpeedMotor2 = MAXSPEED - (MAXSPEED - MINSPEED)/2;
        }
        else
        {
            iSpeedMotor1 = MAXSPEED;
            iSpeedMotor2 = MAXSPEED;
        }
    }
    else
    {
        iSpeedMotor1 = 0;
        iSpeedMotor2 = 0;
    }

    if (iMQTTCross != 0)
    {
        if (iMQTTCross < 0)
        {
            if(iSpeedMotor1 == MAXSPEED)
            {
                if(abs(iMQTTCross) == 1)
                {
                    iSpeedMotor1 = MAXSPEED - (MAXSPEED - MINSPEED)/2;
                }
                else
                {
                    iSpeedMotor1 = MINSPEED;
                }
            }
        }
    }
}

```

```

    }
    else
    {
        if(abs(iMQTTCross) == 1)
            iSpeedMotor1 = MINSPEED;
        else
            iSpeedMotor1 = 0;
    }
    Serial.println("New Speed motor 1: " + String(iSpeedMotor1));
}
else
{
    if(iSpeedMotor2 == MAXSPEED)
    {
        if(abs(iMQTTCross) == 1)
            iSpeedMotor2 = MAXSPEED - (MAXSPEED - MINSPEED)/2;
        else
            iSpeedMotor2 = MINSPEED;
    }
    else
    {
        if(abs(iMQTTCross) == 1)
            iSpeedMotor2 = MINSPEED;
        else
            iSpeedMotor2 = 0;
    }
    Serial.println("New Speed motor 2: " + String(iSpeedMotor2));
}
}
//Write speed to motor pwm
ledcWrite(motor1channel, iSpeedMotor1);
ledcWrite(motor2channel, iSpeedMotor2);
bUpdateMovement = false; //New movement set
}

}

/*
=====
Function:  setupWifi
Returns:  void
Description: Setup wifi to connect to network
=====
*/
void setupWifi()
{
    Serial.println("Connection to: " + String(STASSID));
    WiFi.mode(WIFI_STA);
    WiFi.begin(STASSID, STAPSK);
    while (WiFi.status() != WL_CONNECTED)
    {
        delay(500);
        Serial.print(".");
    }
}

```

```

}
Serial.println("");
Serial.println("WiFi connected");
Serial.println("IP address: ");
Serial.println(WiFi.localIP());
}

/*
=====
Function:  callback
Returns:  void
Description: Will automatical called, if a subscribed topic
             has a new message
topic:     Returns the topic, from where a new msg comes from
payload:   The message from the topic
length:    Length of the msg, important to get content
=====
*/
void callback(char* topic, byte* payload, unsigned int length)
{
    String strMessage = "";
    writeAdvanceDiag("Message arrived from topic: " + String(topic), true);
    writeAdvanceDiag("Message length: " + String(length), true);
    for (int i = 0; i < length; i++)
        strMessage += String((char)payload[i]);
    writeAdvanceDiag("Message is: " + strMessage, true);
    if (String(topic) == String(SubTopicStraight))
    {
        iMQTTStraightNew = strMessage.toInt();
    }
    else if (String(topic) == String(SubTopicCross))
    {
        iMQTTCrossNew = strMessage.toInt();
    }
}

/*
=====
Function:  reconnectMQTT
Returns:  void
Description: If there is no connection to MQTT, this function is
             called. In addition, the desired topic is registered.
=====
*/
void reconnectMQTT()
{
    while (!mqttClient.connected())
    {
        writeAdvanceDiag("Login to MQTT-Broker", true);
        if (mqttClient.connect(clientID.c_str()))
        {
            Serial.println("Connected to MQTT-Broker " + String(MQTT_BROKER));
            writeAdvanceDiag("Subscribe topic " + String(SubTopicStraight) + "", true);

```

```

    mqttClient.subscribe(SubTopicStraight, 1); //Subscribe topic "SubTopicStraight"
    writeAdvanceDiag("Subscribe topic " + String(SubTopicCross) + "", true);
    mqttClient.subscribe(SubTopicCross, 1); //Subscribe topic "SubTopicCross"
}
else
{
    writeAdvanceDiag("Failed with rc=" + String(mqttClient.state()), true);
    Serial.println("Next MQTT-Connect in 3 sec");
    delay(3000);
}
}
}
}

```

```

/*
=====
Function:  writeAdvanceDiag
Returns:   void
Description: Writes advance msg to serial monitor, if
            ADVANCEDIAG >= 1
msg:       Message for the serial monitor
newLine:   Message with linebreak (true)
=====
*/

```

```

void writeAdvanceDiag(String msg, bool newLine)
{
    if (bool(ADVANCEDIAG)) //Check if advance diag is enabled
    {
        if (newLine)
            Serial.println(msg);
        else
            Serial.print(msg);
    }
}

```

```

/*
=====
Function:  DebounceStraight
Returns:   void
Description: Set new value, if debounce is over
            If there is a new valid bUpdateMovement
            will set true
=====
*/

```

```

void DebounceStraight()
{
    if (iMQTTStraightNew != iMQTTStraightLast)
        ulLastDebounceTimeStraight = millis();

    if ((millis() - ulLastDebounceTimeStraight) > ulDebounce)
    {
        if (iMQTTStraightNew != iMQTTStraight)

```

```

    {
        iMQTTStraight = iMQTTStraightNew;
        writeAdvanceDiag("New straight value " + String(iMQTTStraight), true);
        bUpdateMovement = true;
    }
}
iMQTTStraightLast = iMQTTStraightNew;

}

/*
=====
Function:   DebounceCross
Returns:    void
Description: Set new value, if debounce is over
             If there is a new valid bUpdateMovement
             will set true
=====
*/
void DebounceCross()
{
    if (iMQTTCrossNew != iMQTTCrossLast)
        ulLastDebounceTimeCross = millis();

    if ((millis() - ulLastDebounceTimeCross) > ulDebounce)
    {
        if (iMQTTCrossNew != iMQTTCross)
        {
            iMQTTCross = iMQTTCrossNew;
            writeAdvanceDiag("New cross value " + String(iMQTTCross), true);
            bUpdateMovement = true;
        }
    }
    iMQTTCrossLast = iMQTTCrossNew;
}

```

*Code 3: Quellcode vom Rollroboter*

Das Grundprinzip vom Rollroboter ist fast analog zum Arduino mit LCD-Display aus Teil 2. Der Rollroboter initialisiert sich und verbindet sich mit dem WLAN. Im nächsten Schritt verbindet sich der Rollroboter mit dem MQTT-Broker und abonniert die für ihn relevanten Topics. Ändert sich ein Wert der Topics, wird die callback-Funktion aufgerufen und der neue Wert übernommen.

Im Fall vom Rollroboter, dieser meldet sich beim MQTT-Broker als AZbot an, wird der Wert für die Vorwärts- und Seitwärtsbewegung mit den Funktionen „DebounceStraight“ und „DebounceCross“ noch entprellt. Erhalten wir allerdings schnelle Signalsprünge von der Fernbedienung, reagiert der Code nicht bei jedem Zyklus auf die Änderung.

Zuletzt wird in der loop-Funktion der entsprechende Wert der Fernbedienung übernommen und die entsprechenden PWM-Signale gesetzt.

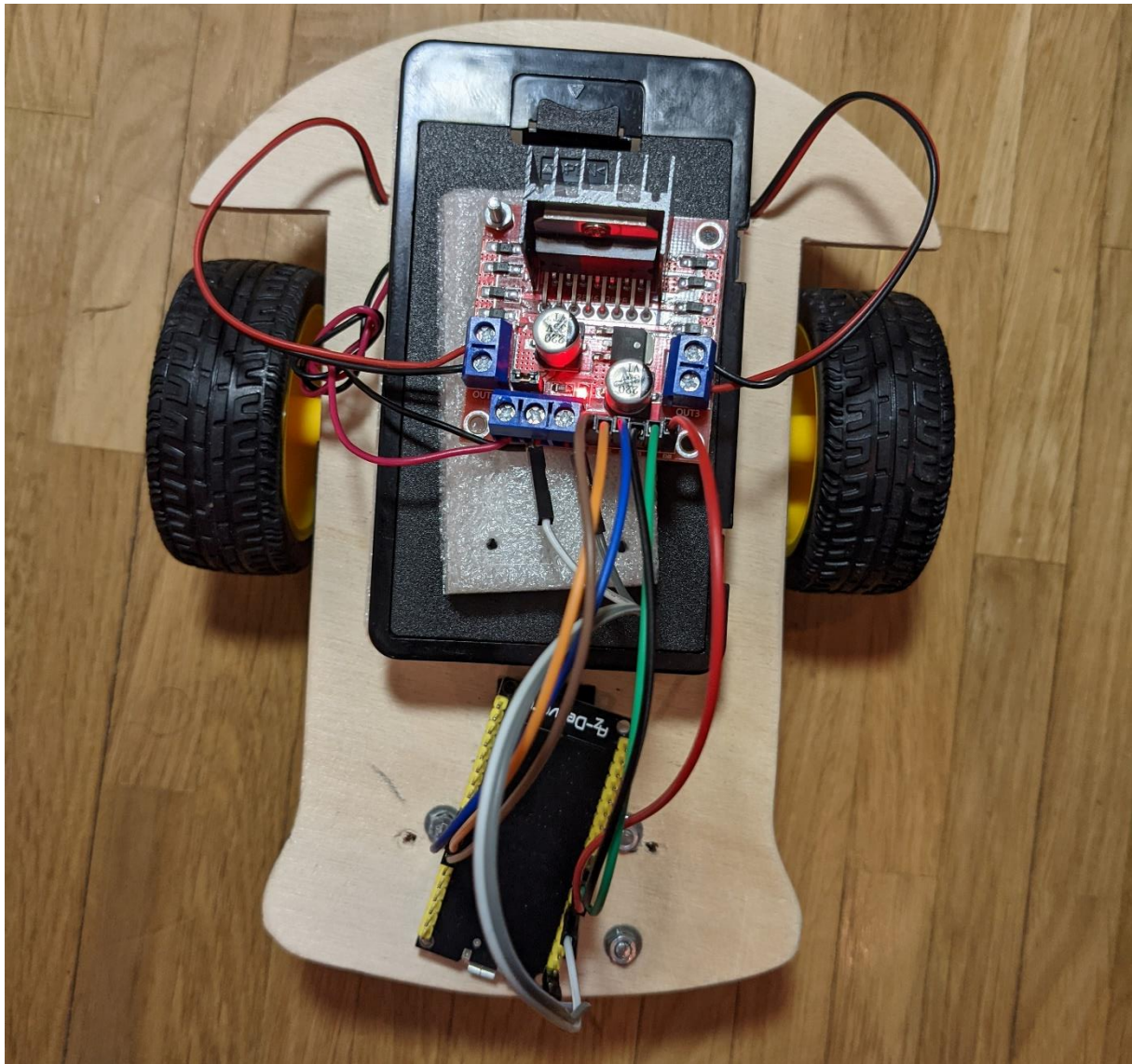


Abbildung 5: Der kleine AzBot darf fahren

Der gezeigte Rollroboter, samt Fernbedienung ist simple. Modifizieren kann man an diesem Grundgerüst an mehreren Stellen. Gerade bei der Fernbedienung würde sich z.B. ein Joystick-Modul geradezu anbieten. Auch der Quellcode kann noch um einiges optimiert werden, ist jedoch für Einsteiger einfach gehalten.

Auch beim Rollroboter sind verschiedenste Modifikationen denkbar. In diesem Beispiel wird die Richtungsänderung nach links oder rechts über das Reduzieren der Geschwindigkeit von einem Motor realisiert. Hier könnte man sich überlegen, das vordere Rad mit einem Servomotor zu verbinden, um besser nach links oder rechts steuern zu können. Kinderfreundlicher wird der Roboter z.B. wenn noch zwei OLED-Displays mit Augen montiert werden und ggf. ein Buzzer als Hupe. Sie sehen, der Anfang für ein weitaus größere Projekt ist getan, nun entscheiden Sie, wie und was ihr Rollroboter alles können soll.

Dieses und weitere Projekte finden sich auf GitHub unter <https://github.com/M3taKn1ght/Blog-Repo>