ERSTSEMESTERPROJEKT

FINALE ABGABE

Finale Abgabe bis zum 22.01.2025, FB03, Gruppe 02-H:

Muhammad Bagier Alaydrus (1574337)
Eren Cicekli (1573864)
Mitansh Morwale (1604520)
Nursaktyo Suryowibowo (1614466)

Inhaltverzeichnis

1. Einleitung 3
   1.1. Finalen Abnahme 3
2. Aufgabenverteilung und Arbeitsweise in der Gruppe 4
   2.1. Mechanischer Aufbau 4
   2.2. Schußmechanismus 4
   2.4. 3D-Druck 5
   2.5. Verkabelung und Löten 5
   2.6. Codierung in der Arduino IDE 6
3. Beschreibung der Lösung 6
   3.1. Probleme beim mechanischen Aufbau und 3D-Bauteilen 6
   3.2. Probleme bei der Codierung 7
   3.3. Probleme bei der Schußmechanismus 7
   3.4 Programmierung des Autos 7
      1. Einbindung von Bibliotheken 7
      2. RemoteXY Konfiguration 7
      3. Pins für Motoren, Sensoren und Servo 8
      4. Setup-Funktion 8
      5. Hauptschleife (loop) 8
      6. Motorsteuerungsfunktionen 8
      7. Schussmechanismus 9
   3.5. Finaler Schaltplan 9
   3.6. FInaler Zustand 10
4. Reflexion 10
5. Nachbesserung 10
6. Zusammenfassung 10
7. Finaler Code zur Steurung des Autos 11

1. Einleitung

Im Wintersemester 2024 erhielten Studierende der Fakultät FB03, nämlich Elektrotechnik und Informatik, das Erstsemesterprojekt. Unsere Projektarbeit besteht darin, einen ferngesteuerten Roboter zu konstruieren und zu programmieren, der in der Lage ist, einen Ball aufzunehmen und zu transportieren. Der Roboter kann den Ball entweder über eine Linie befördern oder ihn schießen. Sobald die Linie erfolgreich überquert ist, stoppt der Roboter automatisch. Dieser Prozess wird in drei Meilensteinen umgesetzt. Das Hauptziel des Projekts besteht darin, die Grundlagen der Informatik in Kombination mit mechanischem Aufbau und Software zu üben und zu vertiefen.

1.1. Finalen Abnahme

Im Rahmen unseres Erstsemesterprojekts haben wir die Entwicklung eines Roboters erfolgreich umgesetzt, indem wir die gestellten Anforderungen schrittweise erreicht haben. Das Projekt war in mehrere Meilensteine unterteilt, die jeweils spezifische Ziele enthielten und aufeinander aufbauten.

- **Meilenstein A**: Bis zum ersten Sprechstundentermin, wurde die Hardware des Roboters zusammengebaut und die Fernsteuerung realisiert. Zudem wurde ein Konzept zum Schießen des Balls entwickelt und die Struktur der Dokumentation festgelegt. Unsere aktuelle Dokumentation wurde termingerecht auf Moodle hochgeladen.
- **Meilenstein B**: Zum zweiten Sprechstundentermin, war der Roboter in der Lage, eigenständig eine Linie zu erkennen und anzuhalten sowie automatisch auf das eigene Spielfeld zurückzukehren. Der Mechanismus zum Schießen des Balls wurde erfolgreich realisiert und die Fortschritte wurden entsprechend dokumentiert.
- **Meilenstein C**: Bis zum dritten Sprechstundentermin, konnte der Roboter ferngesteuert einen Ball aufnehmen, ihn über eine Linie transportieren und danach selbstständig anhalten sowie auf das Startfeld zurückkehren. Die Dokumentation wurde finalisiert und vervollständigt.
- **Finale Abnahme**: Sollte der Roboter für die Teilnahme am Wettbewerb vollständig funktionstüchtig sein.

Diese strukturierte Vorgehensweise ermöglichte es uns, die gestellten Herausforderungen erfolgreich zu meistern und die angestrebten Lernziele zu erreichen.

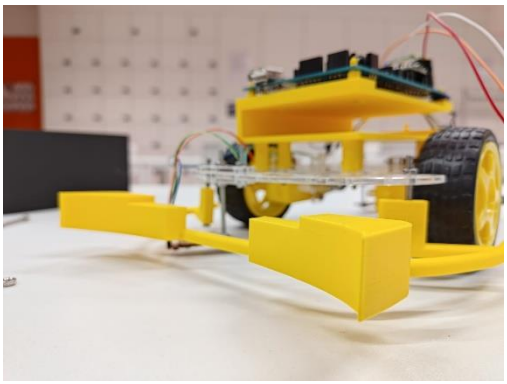
2. Aufgabenverteilung und Arbeitsweise in der Gruppe

Um dieses Projekt zu starten, wird den Studierenden empfohlen, zunächst an Workshops und Sicherheitsunterweisungen teilzunehmen, um Hindernissen vorzubeugen und den Zugang zum Projekterfolg zu erleichtern. Alle Mitglieder dieser Gruppe haben diese Schritte bereits unternommen. Alle Aufgaben und Arbeitspakete, die wir erstellen, basieren auf den Informationsquellen, die in Moodle zur Verfügung gestellt werden.

Daher haben wir einige Aufgaben auf verschiedene Mitglieder verteilt:

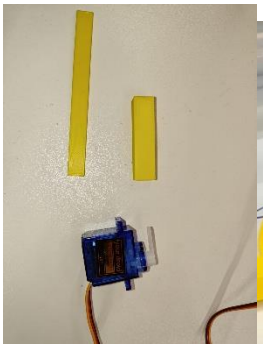
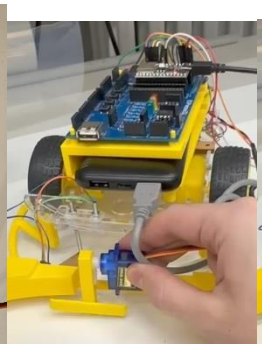
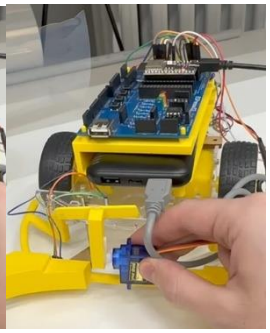
2.1. Mechanischer Aufbau

In diesem Abschnitt wurden die Bauteile wie die beiden Motoren, die Räder und das Schwenkrad mithilfe von Schrauben und Muttern an der Plattform aus Plexiglas montiert. Sakty und Mitansh, zwei unserer Gruppenmitglieder, haben hier tatkräftig unterstützt, indem sie alle Kabel verbunden und weitere Hardwareteile wie den Sensor befestigt haben. Zusätzlich haben wir eine 3D-Design-Stoßstange angebracht, die als Transportvorrichtung für den Ball dient und es ermöglicht, ihn überall hin zu bewegen. Diese Stoßstange wurde speziell mit einer Öffnung in der Mitte entworfen, um Platz für den Schussmechanismus zu schaffen.



2.2. Schußmechanismus

Für den Schussmechanismus haben wir den Mikroservo SG90 verwendet. An den Servoarm wurde ein kleines 3D-Design befestigt, das in seiner Form einem Eisstiel ähnelt. Dieses Bauteil wurde so gestaltet, dass es wie ein Bulldozer-Element wirkt, welches den Ball nach vorne schieben kann. Die präzise Anordnung des Servos und des 3D-Teils ermöglicht eine effektive Schussfunktion, die direkt mit der Steuerung des Roboters verknüpft ist.

2.4. 3D-Druck

Bevor wir größere Änderungen am Design vornehmen konnten, haben wir verschiedene Komponenten wie den Powerbank-Halter, den Platz für die Platine und den Schalter in 3D gedruckt. Sakty und Mitansh haben den 3D-Drucker benutzt, um den Powerbank-Halter erfolgreich herzustellen.

Der Fokus unseres aktuellen 3D-Drucks liegt jedoch auf der Stoßstange an der Vorderseite des Autos, die es ermöglichen soll, den Ball in jede Richtung zu transportieren. Außerdem haben wir versucht, eine kleine Ballkachel für den Schussmechanismus zu drucken, die wir später am Servo befestigen wollen. Bei der Stoßstange selbst stoßen wir jedoch auf Probleme, da uns die nötige Erfahrung und Fähigkeiten im Umgang mit 3D-Design-Software fehlen.

2.5. Verkabelung und Löten

In diesem Abschnitt mussten wir verschiedene Aufgaben im Bereich Verkabelung und Löten bewältigen. Zunächst haben wir die QTR-1A-Sensoren mit den entsprechenden Kabelkontakten verlötet. Danach entfernten wir das Ground- und 5V-Kabel von der USB-Powerbank, um es an den Schalter anzuschließen. Ein weiteres kleines Problem bestand darin, die drei Kabel (Ground von der Powerbank, Platine und H-Brücke) korrekt miteinander zu verbinden. Eren und Bagier waren für das Verlöten der Sensoren verantwortlich.

Für die zusätzlichen elektronischen Komponenten, insbesondere den Mikroservo SG90, mussten wir eine Verbindung zwischen dem Ground- und 5V-Port sowie dem GPIO-Pin herstellen. Ein neues Problem entstand jedoch, da alle 5V-Ports auf unserem Mikrocontroller bereits durch die beiden QTR-1A-Sensoren belegt waren.

Ein großes Problem während des Projekts war das Kabelmanagement. Um die Kabel ordentlich zu halten, haben wir viel Klebeband und Kleber verwendet, damit sie an ihrem Platz bleiben. Außerdem haben wir die Kabel markiert, basierend auf ihrer Quelle und ihrem Ziel, um Verwechslungen zu vermeiden. Besonders wichtig war es, eine stabile Verbindung zwischen der Powerbank, dem Mikrocontroller und den Motoren des Autos sicherzustellen, um Störungen und Ausfälle zu verhindern.

2.6. Codierung in der Arduino IDE

Während der Durchführung der oben genannten Schritte war es notwendig, alle elektronischen Komponenten zu testen, indem wir den Mikrocontroller programmiert haben. Dies geschah, um potenziellen Problemen vorzubeugen. Bagier und Eren waren verantwortlich für die Implementierung des Codes für die Sensoren und das anschließende Testen der Sensorwerte.

Für die Erkennung der schwarzen Linien haben wir eine Schwelle von 4000 definiert. Die Rückgabewerte der beiden QTR-1A-Sensoren liegen bei der Erkennung schwarzer Linien typischerweise bei etwa 4095. In unserem Code wurde programmiert, dass der Roboter stoppt und automatisch für einige Millisekunden zurückfährt oder in eine andere Richtung lenkt, sobald die Sensoren Werte oberhalb der Schwelle von 4000 erkennen.

Für den Schussmechanismus haben wir eine zusätzliche Codefunktion integriert, die den Mikroservo SG90 steuert. Dabei wird der Servo durch die Eingabe von "High" und "Low"-Signalen programmiert, die von der Taste „Button 1“ der RemoteXY-Schnittstelle auf dem Smartphone ausgelöst werden. Button 1 fungiert dabei als Schuss- oder Kick-Taste. Wenn diese Taste gedrückt wird, aktiviert sie den Schussmechanismus und sorgt dafür, dass der Servo die entsprechende Bewegung ausführt, um den Ball nach vorne zu stoßen.

3. Beschreibung der Lösung

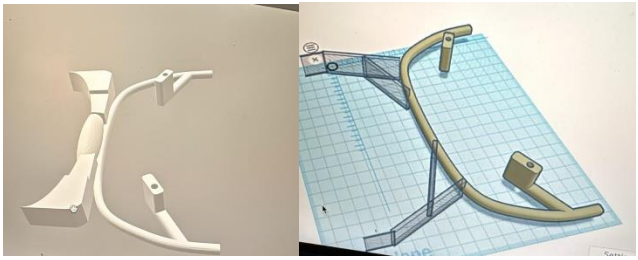
In diesem Abschnitt erklären wir die verschiedenen Probleme, mit denen wir bei jedem der oben genannten Arbeitspakete konfrontiert waren, und erläutern auch, wie wir diese lösen konnten.

3.1. Probleme beim mechanischen Aufbau und 3D-Bauteilen

Wir haben erfolgreich eine Platine und einen Powerbank-Halter in 3D aus einem Design gedruckt, das mit ein paar Optimierungen bereits in Moodle verfügbar ist. Aber wir haben einige Probleme festgestellt. Beim Löten des Schalters hatten Bagier und Eren das Problem, dass der Schalter aufgrund zu hohen Heizens kaputt ging. Daher hat Eren einen neuen Schalter von unserem Professor erhalten.

Erstens passte das Loch in der 3D-Plattform nicht für die Schraube, sodass sie am Plexiglas fest bleiben konnte. Zweitens war das Loch für den Schalter zu klein. Die erste Lösung, die wir fanden, bestand darin, ein Loch in die Oberseite des 3D-Bauteils zu bohren, damit die Schraube hineinpassen konnte. Die Lösung für das zweite Hindernis bestand darin, dass wir das Loch für den Schalter 10 Minuten lang mit einer Feile schleifen mussten, damit es breiter werden konnte.

Um die Stoßstange zu verbessern, haben wir beschlossen, einen zweiten Versuch zu unternehmen. Dieses Mal soll die Stoßstange eine leichte Kurve oder eine dreieckige Form haben, damit der Ball beim Einfangen automatisch zur Mitte gelenkt wird. Unten sind die Bilder beider Designs dargestellt, um den Entwicklungsprozess zu verdeutlichen.



3.2. Probleme bei der Codierung

Zuerst hatten wir ein einfaches Problem, nämlich dass wir den Port (COM1) in der Arduino IDE nicht auswählen konnten. Es stellte sich heraus, dass wir ein Ladekabel verwendet hatten, um den Mikrocontroller und den Laptop zu verbinden, obwohl wir ein Datenkabel hätten verwenden müssen. Danach konnte der kompilierte Code erfolgreich hochgeladen werden.

Abgesehen davon hatten wir auch zu Beginn der RemoteXY-Konfiguration Probleme wie fehlende Dateien und Bibliotheken. Wir mussten die Bibliothek/Zip-Datei für RemoteXY einrichten.

3.3. Probleme bei der Schußmechanismus

Allerdings gibt es ein entscheidendes Problem, das uns weiterhin beschäftigt: Die Leistung des Mikroservos SG90 ist für den Schussmechanismus unzureichend. Zwar kann der Servo den Ball stoßen, jedoch fehlt ihm die notwendige Kraft, um den Ball direkt und präzise ins Ziel zu befördern. Dies stellt eine Einschränkung dar, die wir in der nächsten Projektphase angehen müssen.

3.4 Programmierung des Autos

1. Einbindung von Bibliotheken

Die Bibliotheken `ESP32Servo.h`, `BLEDevice.h` und `RemoteXY.h` werden eingebunden, um die Servo-Steuerung, Bluetooth-Konnektivität und RemoteXY-Bedienoberfläche zu ermöglichen. Die RemoteXY-Bibliothek wird für die einfache Erstellung von mobilen Benutzeroberflächen zur Steuerung von IoT-Geräten verwendet.

2. RemoteXY Konfiguration

Der Modus `REMOTEXY_MODE__ESP32CORE_BLE` wird definiert, um Bluetooth-Konnektivität über das ESP32-Modul zu nutzen. Die Parameter `REMOTEXY_BLUETOOTH_NAME` und `REMOTEXY_ACCESS_PASSWORD` legen den Bluetooth-Namen und ein Zugangspasswort für die Verbindung fest. Eine RemoteXY-Konfiguration wird definiert und enthält Bedienelemente wie Joystick, Tasten und LEDs, die über eine kompakte Binärstruktur verwaltet werden.

3. Pins für Motoren, Sensoren und Servo

Die Pins für die Motoren (`Motor_links_A`, `Motor_links_B`, `Motor_rechts_A`, `Motor_rechts_B`), die analogen Sensoren (`frontSensorPin`, `rearSensorPin`) und den Servo (`servoPin`) werden definiert. Zusätzlich werden Variablen für die Verarbeitung der Sensordaten und Schwellwerte (`threshold`) sowie den Zustand einer Schaltfläche (`previousButton1State`) initialisiert

4. Setup-Funktion

In der `setup`-Funktion wird die RemoteXY-Bibliothek initialisiert, die Baudrate der seriellen Kommunikation festgelegt und die Pins für Motoren und Sensoren als Ein- oder Ausgänge konfiguriert. Der Servo wird an den definierten Pin angeschlossen, in die Startposition bewegt und die Motoren werden auf einen Stoppzustand gesetzt.

5. Hauptschleife (`loop`)

Die `loop`-Funktion enthält die Hauptlogik des Programms:

- **Sensorwerte lesen**: Die Werte der Front- und Rücksensoren werden eingelesen und zur Analyse verwendet.
- **Linienerkennung**: Wenn ein Sensor einen bestimmten Schwellenwert überschreitet, wird eine Bewegung wie Rückwärtsfahren (`autoBack`) oder Vorwärtsfahren (`moveForward`) ausgelöst.
- **Joystick-Steuerung**: Die Motoren werden basierend auf den Eingaben des RemoteXY-Joysticks gesteuert.
- **Schussmechanismus**: Ein Tastenbefehl löst die Bewegung eines Servos aus, der für einen Schussvorgang zuständig ist.

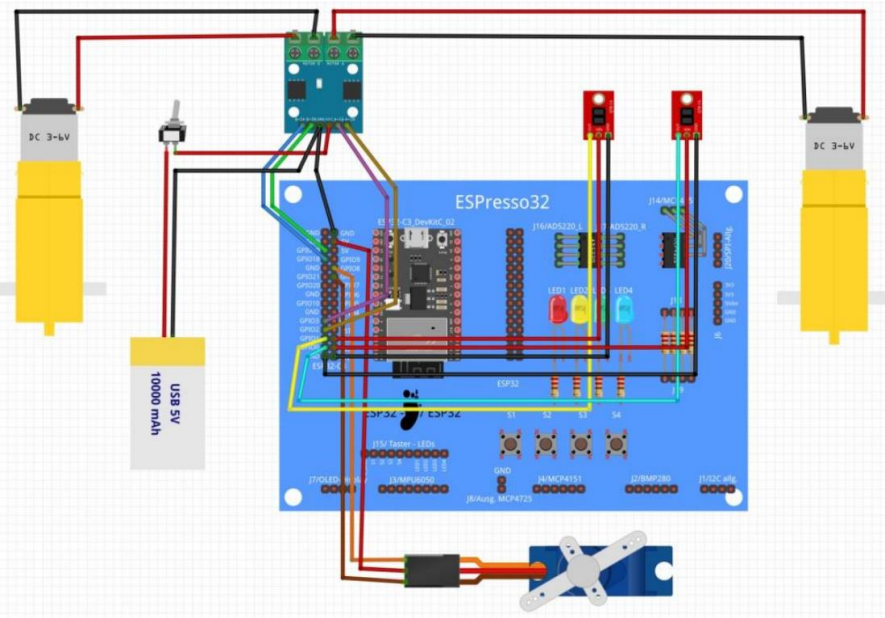
6. Motorsteuerungsfunktionen

- **`stopMotors`**: Stoppt alle Motoren, indem die Pins auf `LOW` gesetzt werden.
- **`autoBack`**: Lässt das Fahrzeug kurz rückwärts fahren.
- **`moveForward`**: Lässt das Fahrzeug kurz vorwärts fahren.
- **`controlMotorsWithJoystick`**: Berechnet die Geschwindigkeiten der linken und rechten Motoren basierend auf Joystick-Eingaben, um lineare Bewegung und Drehungen zu ermöglichen.

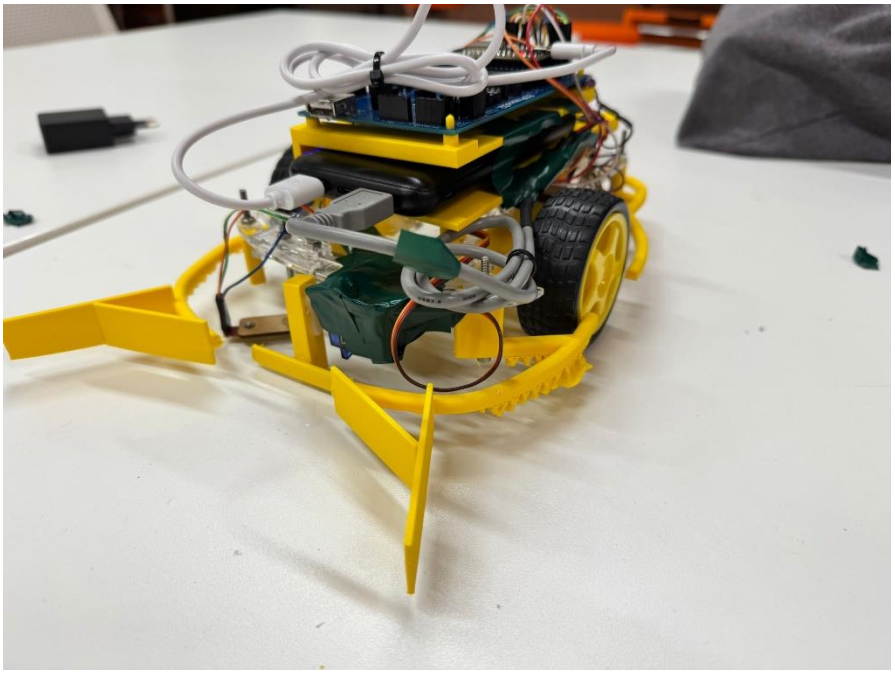
7. Schussmechanismus

Die Funktion `shootBall` steuert den Servo, um eine Schussbewegung auszuführen. Der Servo bewegt sich zur Schussposition (`180 Grad`), hält diese kurz und kehrt dann in die Ausgangsposition (`0 Grad`) zurück.

3.5. Finaler Schaltplan



3.6. FInaler Zustand



4. Reflexion

Beide Sensoren funktionieren einwandfrei, und wir können unser Projekt erfolgreich über die App fernsteuern. Zu Beginn hatten wir ein Problem mit dem Powerbankhalter, welches wir jedoch erfolgreich beheben konnten. Auch die anderen Mechanismen, die wir in diesem Projekt implementiert haben, arbeiten größtenteils zuverlässig und entsprechen unseren Erwartungen.

5. Nachbesserung

Während dieses Projekts haben wir viele Herausforderungen gemeistert, die uns wertvolle Lektionen gelehrt haben. Besonders schwierig waren die mechanischen und elektronischen Komponenten, insbesondere der Umgang mit dem ESP32C-Mikrocontroller und die Verbindung der verschiedenen Bauteile. Gleichzeitig hatten wir große Defizite im Bereich des Designs. Unser Fokus lag ausschließlich darauf, das Auto funktional zu machen, wodurch das ästhetische Erscheinungsbild komplett vernachlässigt wurde. Für zukünftige Projekte würden wir uns vornehmen, nicht nur funktionale Lösungen zu finden, sondern auch kreativere Designs zu entwickeln. Wir glauben, dass wir durch den bewussten Einsatz von Kreativität ein beeindruckenderes und durchdachteres Produkt schaffen könnten. Außerdem möchten wir uns intensiver mit Design und Gestaltung beschäftigen, um das Auto nicht nur funktionsfähig, sondern auch optisch ansprechend zu gestalten.

6. Zusammenfassung

Im Rahmen des Projekts haben wir erfolgreich den mechanischen Aufbau des Roboters abgeschlossen, einschließlich der Installation von Motoren, Rädern und dem Schwenkrad. Zudem wurden wichtige

Komponenten wie der Powerbank-Halter und der Platz für die Platine mithilfe von 3D-Druck erstellt. Es gab jedoch einige Herausforderungen, wie die fehlerhafte Größe der Löcher im 3D-gedruckten Bauteil und das Überhitzen des Schalters beim Löten, die wir durch Nachbearbeitung und Austausch des Schalters beheben konnten. Bei der Verkabelung und der Codierung traten ebenfalls kleinere Probleme auf, zum Beispiel bei der Auswahl des richtigen Kabels und bei fehlenden Bibliotheken für die RemoteXY-App, die wir jedoch schnell lösen konnten. Beide Sensoren arbeiten nun einwandfrei, und der Roboter lässt sich erfolgreich über die App fernsteuern.

Zusätzlich zu den oben genannten Punkten haben wir den Roboter mit einem 3D-gedruckten Stoßfänger ausgestattet, um den Ball transportieren zu können, und eine Öffnung für den Schussmechanismus integriert. Trotz erster fehlerhafter Designs konnten wir den Stoßfänger verbessern, um den Ball effizient zur Mitte zu führen. Für den Schussmechanismus haben wir einen Mikroservo SG90 verwendet, der jedoch nicht genug Kraft bietet, um den Ball präzise ins Ziel zu schießen. Als Alternative erwägen wir einen leistungsstärkeren Servo oder einen Solenoid. Insgesamt konnten wir die meisten Mechanismen erfolgreich umsetzen, müssen jedoch die Schussleistung weiter optimieren.

7. Finaler Code zur Steurung des Autos

```
#include <ESP32Servo.h>

////////////////////////////////////////////
// RemoteXY-Bibliothek einbinden //
////////////////////////////////////////////
```

```
#define REMOTEXY_MODE__ESP32CORE_BLE
#include <BLEDevice.h>
#include <RemoteXY.h>

// RemoteXY-Verbindungseinstellungen
#define REMOTEXY_BLUETOOTH_NAME "CyberTruckLite"
#define REMOTEXY_ACCESS_PASSWORD "2004"

// RemoteXY-Konfiguration
#pragma pack(push, 1)
uint8_t RemoteXY_CONF[] =
{
  255,5,0,3,0,62,0,16,200,0,5,32,2,24,37,37,31,213,31,10,
  48,48,47,10,10,4,36,31,79,78,0,31,79,70,70,0,1,8,79,41,
  13,13,213,31,75,105,99,107,0,1,8,49,33,9,9,213,31,82,101,115,
  101,116,0,65,112,91,1,7,7
};

struct {
  int8_t joystick_x; // von -100 bis 100
  int8_t joystick_y; // von -100 bis 100
  uint8_t pushSwitch_1; // =1, wenn Zustand EIN ist, sonst =0
  uint8_t button_1; // =1, wenn Knopf gedrückt, sonst =0
  uint8_t button_2; // =1, wenn Knopf gedrückt, sonst =0
  uint8_t led_2_r; // =0..255 LED-Rot-Helligkeit
  uint8_t led_2_g; // =0..255 LED-Grün-Helligkeit
  uint8_t led_2_b; // =0..255 LED-Blau-Helligkeit
  uint8_t connect_flag; // =1, wenn Verbindung hergestellt, sonst =0
} RemoteXY;
#pragma pack(pop)

////////////////////////////////////////////
// Motor-, Sensor- und Servo-Pins //
////////////////////////////////////////////
int Motor_links_A = 3;  // Pin für Motor links A
int Motor_links_B = 2;  // Pin für Motor links B
int Motor_rechts_A = 18; // Pin für Motor rechts A
int Motor_rechts_B = 19; // Pin für Motor rechts B

int frontSensorPin = 0;  // Analogpin für QTR-1A-Sensor (vorne)
int rearSensorPin = 1;   // Analogpin für QTR-1A-Sensor (hinten)

int frontSensorValue = 0;  // Variable zum Speichern des
Frontsensorwerts
int rearSensorValue = 0;   // Variable zum Speichern des
Rücksensorwerts
```

```
int threshold = 4000;      // Schwellenwert zur Erkennung einer
schwarzen Linie
bool previousButton1State = false;

int servoPin = 5;          // Pin für den SG90-Servo
Servo kickServo;           // Servo-Objekt

////////////////////////////////////////////
// Setup-Funktion //
////////////////////////////////////////////
void setup() {
  RemoteXY_Init();
  Serial.begin(115200);

  // Setze die Motorpins als Ausgänge
  pinMode(Motor_links_A, OUTPUT);
  pinMode(Motor_links_B, OUTPUT);
  pinMode(Motor_rechts_A, OUTPUT);
  pinMode(Motor_rechts_B, OUTPUT);

  // Richten Sie die Sensorpins ein
  pinMode(frontSensorPin, INPUT);
  pinMode(rearSensorPin, INPUT);

  // Initialisiere den Servo
  kickServo.attach(servoPin);
  kickServo.write(0); // Sicherstellen, dass der Servo in der
Ausgangsposition ist

  // Beide Motoren initial anhalten
  stopMotors();
}

////////////////////////////////////////////
// Hauptschleife //
//////////    digitalWrite(Motor_rechts_B, HIGH);
  } else {
    digitalWrite(Motor_rechts_A, HIGH);
    digitalWrite(Motor_rechts_B, LOW);
  }
}

// Funktion zur Steuerung der Schussmechanik
void shootBall() {
  kickServo.write(90); // Servo in Schussposition drehen
  delay(500);          // Position kurz halten
  kickServo.write(0);  // In Ausgangsposition zurückkehren
```

```
} ////////////////////////////////////
void loop() {
  RemoteXY_Handler();

  // Sensorwerte auslesen
  frontSensorValue = analogRead(frontSensorPin);
  rearSensorValue = analogRead(rearSensorPin);

  Serial.print("Frontsensor: ");
  Serial.print(frontSensorValue);
  Serial.print("  Rücksensor: ");
  Serial.println(rearSensorValue);

  // Überprüfen, ob der Frontsensor eine schwarze Linie erkennt
  if (frontSensorValue >= threshold) {
    stopMotors();
    autoBack();
    stopMotors();
  } else if (rearSensorValue >= threshold) {
    stopMotors();
    moveForward();
    stopMotors();
  } else {
    controlMotorsWithJoystick(RemoteXY.joystick_x,
RemoteXY.joystick_y);
  }

  // Überprüfen, ob der Schussbefehl vorliegt
  if (RemoteXY.button_1 == 1 && !previousButton1State) {
    // Nur ausführen, wenn Knopf gedrückt ist und zuvor nicht gedrückt
war
    shootBall();
  }
  // Aktualisieren des vorherigen Zustands des Knopfes
  previousButton1State = (RemoteXY.button_1 == 1);
}

////////////////////////////////////////////
// Motor- und Schussfunktionen //
////////////////////////////////////////////

// Funktion zum Stoppen beider Motoren
void stopMotors() {
  digitalWrite(Motor_links_A, LOW);
  digitalWrite(Motor_links_B, LOW);
  digitalWrite(Motor_rechts_A, LOW);
  digitalWrite(Motor_rechts_B, LOW);
```

```
}

// Funktion, um das Auto rückwärts fahren zu lassen
void autoBack() {
  digitalWrite(Motor_links_A, HIGH);
  digitalWrite(Motor_rechts_A, HIGH);
  delay(143);
}

// Funktion, um das Auto vorwärts fahren zu lassen
void moveForward() {
  digitalWrite(Motor_links_A, LOW);
  digitalWrite(Motor_links_B, HIGH);
  digitalWrite(Motor_rechts_A, LOW);
  digitalWrite(Motor_rechts_B, HIGH);
  delay(143);
}

// Funktion zur Steuerung der Motoren basierend auf Joystick-Eingaben
void controlMotorsWithJoystick(int8_t x, int8_t y) {
  int speedLeft = 0;
  int speedRight = 0;

  if (x == 0 && y == 0) {
    stopMotors();
    return;
  }

  int motorSpeed = map(abs(y), 0, 100, 0, 255);
  int turnSpeed = map(abs(x), 0, 100, 0, 255);

  if (y > 0) {
    speedLeft = motorSpeed;
    speedRight = motorSpeed;
  } else if (y < 0) {
    speedLeft = -motorSpeed;
    speedRight = -motorSpeed;
  }

  if (x < 0) {
    speedLeft += turnSpeed;
    speedRight -= turnSpeed;
  } else if (x > 0) {
    speedLeft -= turnSpeed;
    speedRight += turnSpeed;
  }
```

```
if (speedLeft >= 0) {
  digitalWrite(Motor_links_A, LOW);
  digitalWrite(Motor_links_B, HIGH);
} else {
  digitalWrite(Motor_links_A, HIGH);
  digitalWrite(Motor_links_B, LOW);
}

if (speedRight >= 0) {
  digitalWrite(Motor_rechts_A, LOW);
```