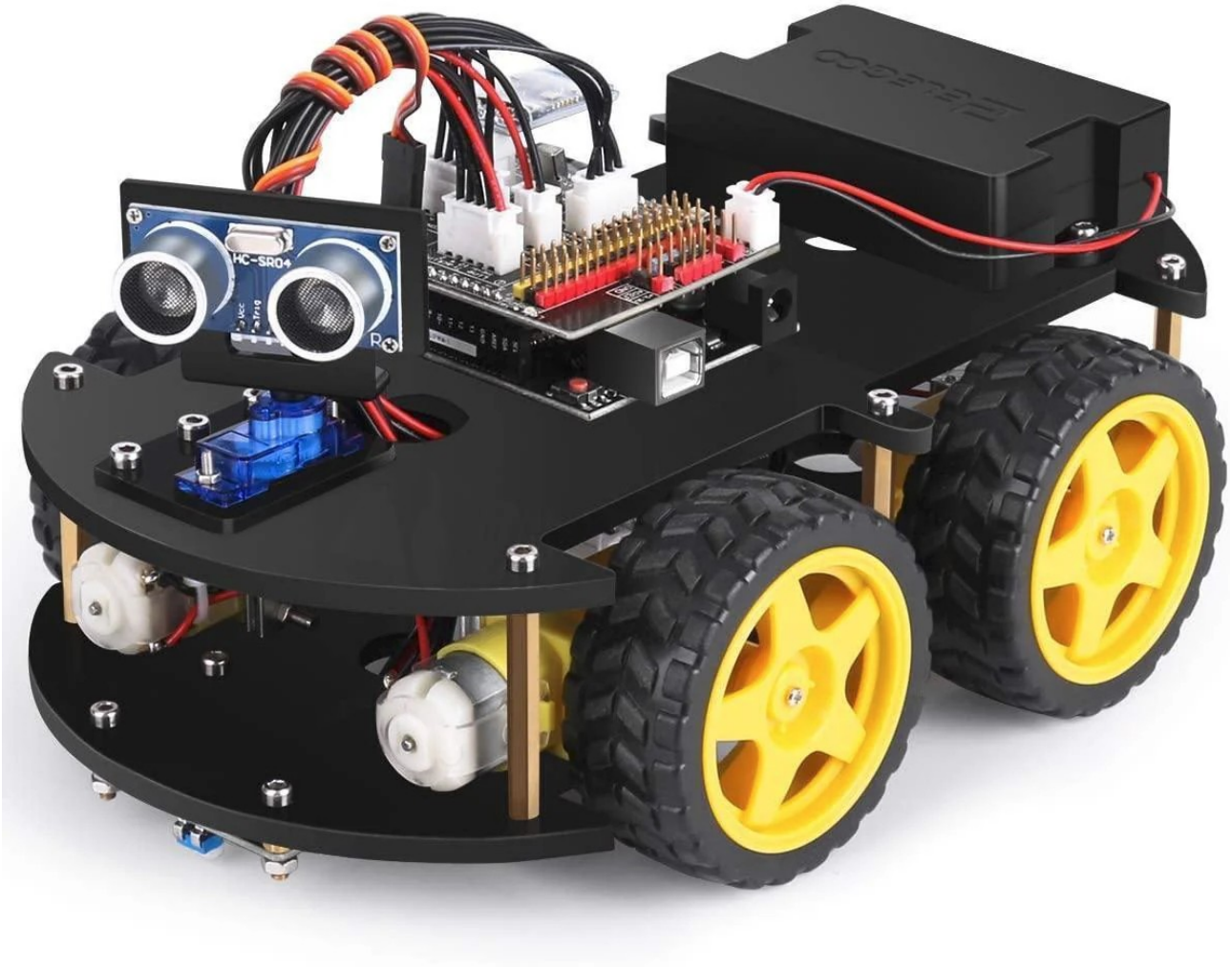
Dokumentation Arduino Car Projekt Gruppe 8



Mitglieder: Jonas Boekhorst und Kai Maier

Einführung und Mitglieder

Dies ist ein Projekt fuer das Fach Rechner und Systemtechnik bei dem Lehrer Herrn. Wißen.

Dies ist die Dokumentation der Gruppe 8, die aus dem Mitgliedern Kai Maier und Jonas Boeckhorst besteht.

Inhaltsangabe

Inhaltsverzeichnis

Übersicht des Projektes

Ziel des Projektes

Das Ziel des Projektes ist, das Elegoo Arduino Car 3.0 mit mehreren Funktionen auszustatten und dies in einer gemeinsamen Codebasis zu vereinen. Dazu gehört das Programmieren, Dokumentieren und das Präsentieren des Projektes.

Die Herausforderung liegt in dem Aufteilen der Aufgaben, sowie eine gute Kommunikation zu zwischen den Mitgliedern zu erstellen und die einzelnen Code-Snippets erfolgreich zusammenzufügen.

Aufgabenverteilung

Aufgaben	Zu Erledigen von	Erledigt von
Ultraschall Sensor Programmieren (Objekt)	Kai	Kai
Infrarot Modul	Kai	Kai
Servo Motor Programmieren (Objekt)	Kai	Kai
H-Brücke Programmieren (Objekt)	Jonas	Jonas/Kai
Bluetooth-Modul Programmieren (Objekt)	Kai	Kai
Linien-Erkennung Programmieren (Objekt)	Kai	Kai
Manuelles Fahren Programmieren	Kai	Kai
Automatisches Fahren Programmieren	Jonas	Jonas
Automatische Linienverfolgung Programmieren	Kai	Kai
Kontrolllogik Programmieren	Kai	Kai
Präsentation Erstellen	Jonas	/
Dokumentation Erstellen	Kai	Kai

Info: Die oben angegebenen Daten sind zum Stand des 23.06.2024.
Info: Die Markierung (Objekt) bedeutet, dass das physische Modul als Objekt in der Software implementiert wird, um die Programmierung und Kontrolle zu vereinfachen.

Aufbau der Klassen

Car
+ mode : String{} + servo_motor : ServoMotor + h_bruecke : Hbridge + line_tracking_modul : LineTracking + on_off : bool + data_array : int** + richtung : String{}
+ Car(Servo) : Car + start : void - automatisches_fahren_mit_ausweichen(int) : void - manuell : void - print_cords : void - deconstruct_data_array(int**) : void - bluetooth_handler : void - infra_red_handler : void

ServoMotor
- servo : Servo - start_pos : int - sonic_sensor : SuperSonic - min_pos : int - max_pos : int - pos : int - data_array : int**
+ set_servo(Servo) : void + get_frontal_distance : int + get_cords(int) : int** + deconstruct_data_array : void

SuperSonic
+ get_distance : int

Hbridge
+ drive_foward(int) : void + drive_backward(int) : void + stop : void + turn_right(int) : void + turn_right_90_degrees(int) : void + turn_left(int) : void + turn_left_backward(int) : void + turn_right_backward(int) : void + turn_left_90_degrees(int) : void + spin_right(int) : void + spin_left(int) : void - motor_left_forward : void - motor_right_forward : void - motor_right_backward : void - motor_left_backward : void - motor_right_stop : void - motor_left_stop : void

LineTracking
- h_bridge : Hbridge
+ LineTracking(Hbridge) : LineTracking + line_tracking : void - get_sensor_vals : void

Software Hierarchie und Code Flow

Der Codeflow ist sehr einfach gehalten.

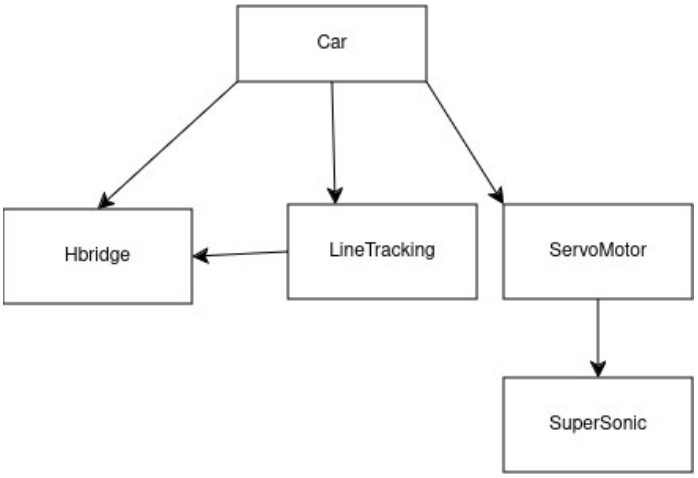
Wir starten im manuellen Modus, mit der aktuellen Aktion „Stop“.

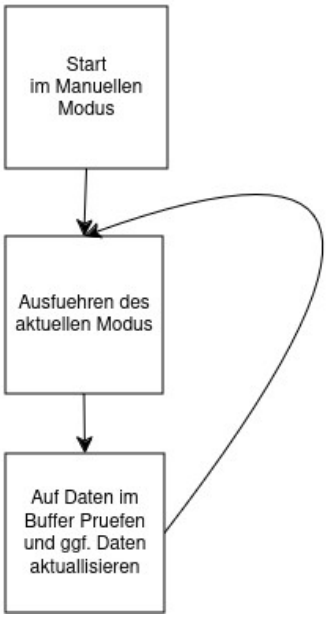
Wenn wir nun eine Kommunikation via Bluetooth oder Infrarot aufgebaut haben, können wir das Auto steuern.

Es wird immer der Modus ausgeführt und dann mit dem Bluetooth handler oder dem Infrared Handler auf Veränderungen geprüft.

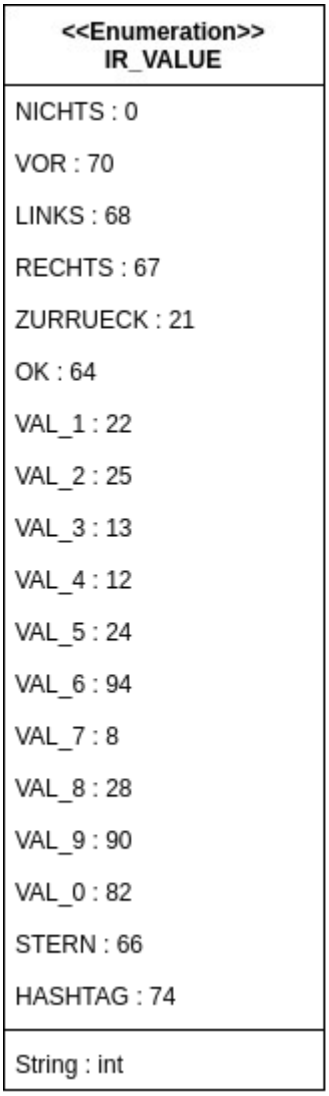
Dies kann beispielsweise ein Modus wechsel sein, oder ein Richtungswechsel beim Manuellen fahren.

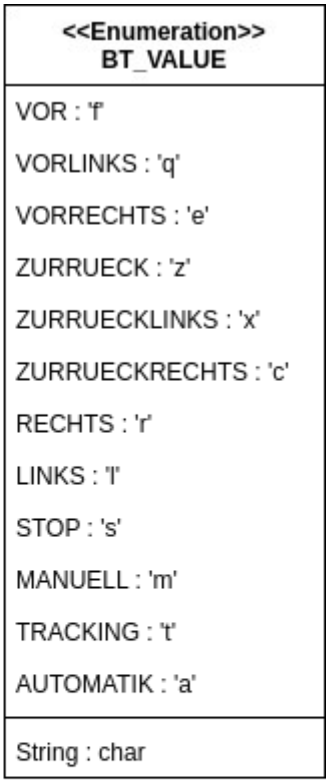
Verzweigung der Klasse





Enum Klassen





Die Enum Klassen werden hauptsächlich zur Vereinfachung des Quellcodes verwendet, damit man einfacher arbeiten kann. Diese werden zurzeit in der Bluetooth Handler Funktion, sowie in der Infra Rot Handler Funktion verwendet.

Verwendete Externe Libraries

Servo.h

Wird als Interface benötigt, um den Servomotor zu steuern. Die selbst erstellte ServoMotor Klasse dient als Wrapper, um das Steuern zu vereinfachen.

Links: Arduino, Github

IRremote.h

Wird als Interface benötigt, um die Kommunikation via Infrarot zu gewährleisten.

Links: Github

Fest definierte Variablen

```
1  #define lt_rechts 10 //Pin Linetracking Rechts
2  #define lt_mitte 4   //Pin Linetracking Mitte
3  #define lt_links 2   //Pin Linetracking Links
4  #define servo_out 3  //Pin Servo Motor
5  #define h_br_en1 6 //Pin Motoren Links an/aus
6  #define h_br_in1 11 //Pin Motor Links S1
7  #define h_br_in2 9 //Pin Motor Links S2
8  #define h_br_in3 8 //Pin Motor Rechts S1
9  #define h_br_in4 7 //Pin Motor Rechts S2
10  #define h_br_en2 5 //Pin Motoren Rechts an/aus
11  #define ul_sonic_trig A5 // Ultrashall Sensor Output
12  #define ul_sonic_echo A4 //Ultraschall Sensor Input
13  #define infra_red 12 // Infrarot Sensor Input
14  #define bt_rx 0 //Bluetooth Receive
15  #define bt_tx 1 // Bluethooth Transmit
```

Setup der Pins

```
1  void setup() {
2     Serial.begin(9600);
3     IrReceiver.begin(infra_red);
4     pinMode(lt_rechts, INPUT_PULLUP);
5     pinMode(lt_mitte, INPUT_PULLUP);
6     pinMode(lt_links, INPUT_PULLUP);
7     pinMode(h_br_en1, OUTPUT);
8     pinMode(h_br_en2, OUTPUT);
9     pinMode(h_br_in1, OUTPUT);
10     pinMode(h_br_in2, OUTPUT);
11     pinMode(h_br_in3, OUTPUT);
12     pinMode(h_br_in4, OUTPUT);
13     pinMode(infra_red, INPUT_PULLUP);
14     pinMode(ul_sonic_echo, INPUT_PULLUP);
15     pinMode(ul_sonic_trig, OUTPUT);
16     servo_motor.attach(servo_out);
17
18  }
19
```

Das Bluetooth Modul hat benötigt eine Baud Rate von 9600 und schreibt in den Seriellen Buffer, wie man es in Zeile 2 sehen kann.

Die Einzelnen Klassen Erklärt

Die SuperSonic Klasse

Die SuperSonic Klasse wird als Schnittstelle für den Ultraschallsensor verwendet.
Die Klasse bietet die Möglichkeit, mit der get_distance Methode den Abstand in mm wiederzugeben.

Diese Klasse kann und sollte nur für Ultraschall Sensoren verwendet werden.

Methoden

Die „get_distance“ Methode

```
1  class SuperSonic{ // Klasse fuer den Ultraschall sensor zum vereinfachen der Steuerung
2
3   public:
4      int get_distance(){
5         // Funktion zum erhalten des Frontralen abstand in mm
6         digitalWrite(ul_sonic_trig, LOW);
7         delayMicroseconds(2);
8         digitalWrite(ul_sonic_trig, HIGH);
9         delayMicroseconds(10);
10         digitalWrite(ul_sonic_trig, LOW);
11         unsigned long duration = pulseIn(ul_sonic_echo, HIGH);
12         return int((duration * 0.0343)/2);
13      }
14
15   };
```

Diese Methode gibt den Abstand eines Objektes zum Ultraschallsensor im mm als int zurück.

Die ServoMotor Klasse

Attribute

```
1  class ServoMotor{ // Klasse zur vereinfachten Steuerung des Servo motors
2    private:
3       Servo servo; // Zugriff auf den Servo motor
4       int start_pos = 90; //Standard position 90 degrees
5       SuperSonic sonic_sensor = SuperSonic(); // Zugriff auf den Ultraschall sensor
6       int min_pos = 30;
7       int max_pos = 150;
8       int pos = 90;
9       int** data_array;
```

Methoden

Die „set_servo“ Methode

```
1 void set_servo(Servo new_servo){
2     // Funktion zum setzen des Servos
3     this->servo = new_servo;
4     servo.write(start_pos); // Standart Position anfahren
5 }
```

Diese Funktion wird Benötigt, um der ServoMotor Klasse einen Servo Motro zuzuweisen

Die ServoMotor Klasse wird dazu verwendet, den Servomotor zu steuern und Hilfsmethoden für den Ultraschallsensor bereitzustellen.

Diese Klasse sollte nur für Servo Motoren verwendet werden, die einen Ultraschallsensor besitzen.

Die „get_frontal_distance“ Methode

```
1  int get_frontal_distance(){
2   // Erhalten des Frontalen abstand, da die get_distance() Funktion Privat ist
3   return this->sonic_sensor.get_distance();
4 }
```

Diese Methode dient dazu, die frontale Distanz vom Ultraschallsensor über die SuperSonic Klasse zu erhalten.

Die „get_cords“ Methode

```
int** get_cords(int delay_time = 20){
  // Erstellen eines 2D Arrays, um die Postionen zu erfassen bsp: [[80,40],[70,44],[60, 55]]
  int entries = 13;
  int step = 0;
  int** data_array = new int*[entries]; // Das grosses Array fuer 12  Eintraege erstellen
  for (int i = 0; i < entries; ++i) {
    data_array[i] = new int[2];       // Innere Arrays Erstellen [[2], [2], [2]...] -> [Grad, Wert]
  }
  if(pos < 90){
    servo.write(30);
    delay(delay_time*3);
    for(pos = 30; pos <= 150; pos++){
      servo.write(pos);
      delay(delay_time);
      if((pos % 10) == 0){
        data_array[step][0] = pos;
        data_array[step][1] = sonic_sensor.get_distance();
        step++;
      }}
   }  else {
    servo.write(150);
    delay(delay_time*3);

    for(pos = 150; pos >= 30; pos--){
      servo.write(pos);
      delay(delay_time);
      if((pos % 10) == 0){
        data_array[step][0] = pos;
        data_array[step][1] = sonic_sensor.get_distance();
        step++;
      }
    }}
  return data_array;
}
```

Diese Methode gibt ein 2D Array zurück, in Form von [[Grad, Abstand],…].

Das äußere Array hat 12 Einträge, die einen Winkel von 30° bis 150° beinhalten, also ein effektiver Kegel von 120°. Bei 0° ist der Servomotor auf den Linken Anschlag und bei 180° auf den Rechten Anschlag.

```
int** data_array = servo_motor.get_cords(); //Data Array mit den Koordinaten erstellen
for (int i = 0; i < 13; ++i) {  //Durchlaufen des Arrays
   Serial.print("Eintrag: "); //Eintrag NR
   Serial.print(i);
   Serial.print(" Degrees: ");
   Serial.print(data_array[i][0]); //Gradzahl
   Serial.print(" Distance: ");
   Serial.print(data_array[i][1]); //Abstand in mm
   Serial.println();
}
deconstruct_data_array(data_array); //Freigeben der Arrays, da man sonst einen Memory leak hat
servo_motor.deconstruct_data_array();
```

Die „deconstruct_data_array“ Methode

```
1    void deconstruct_data_array(){
2      // Funktion zum Freigeben des Speichers nach nutzung des get_Cords Arrays
3      for (int i = 0; i < 13; ++i) {
4        delete[] this->data_array[i];
5      }
6        delete[] data_array;
7      }
```

Eine Methode zum Freigeben des Speichers nach Verwendung der get_cords Methode.

Die Hbridge Klasse

Methoden

Die „drive_forward“ Methode

```
1  void drive_forward(int speed = 200) {
2    //Funktion zum Vorwaertsfahren
3    motor_left_forward();
4    motor_right_forward();
5
6    analogWrite(h_br_en1, speed);
7    analogWrite(h_br_en2, speed * 0.75);
8  }
```

Diese Funktion läßt das Arduino Car vorwärts fahren.

Der Geschwindigkeitsmultiplikator von 0,75 gleicht eine die linke Seite des Autos aus, da die rechte Seite durch defekte Teile langsamer ist.

Die „drive_backward“ Methode

```
1  void drive_backward(int speed = 200) {
2    //Funktion zum Rueckwertsfahren
3    motor_left_backward();
4    motor_right_backward();
5
6    analogWrite(h_br_en1, speed);
7    analogWrite(h_br_en2, speed * 0.75);
8  }
```

Diese Funktion läßt das Arduino Car Rückwärts fahren.

Der Geschwindigkeitsmultiplikator von 0,75 gleicht eine die linke Seite des Autos aus, da die rechte Seite durch defekte Teile langsamer ist.

Die „stop“ Methode

```
1  void stop() {
2    //Funktion zum Stoppen der Motoren
3    analogWrite(h_br_en1, 0);
4    analogWrite(h_br_en2, 0);
5  }
```

Diese Methode dient dazu das Auto zu stoppen, indem es die Geschwindigkeit der Motoren auf 0 setzt.

Die „turn_right“ Mehtode

```
1  void turn_right(int speed = 200) {
2    // Im bogen rechts einbiegen
3    motor_left_forward();
4    motor_right_forward();
5
6    analogWrite(h_br_en1, speed/4);
7    analogWrite(h_br_en2, speed);
8  }
```

Diese Methode erlaubt es dem Auto in einem kleinem Bogen nach rechts abzubiegen.

Das Auto fahrt nach vorne.

Die „turn_left“ Methode

```
1  void turn_left(int speed = 200) {
2    // Im bogen Links einbiegen
3
4    motor_left_forward();
5    motor_right_forward();
6
7    analogWrite(h_br_en1, speed);
8    analogWrite(h_br_en2, speed/4);
9  }
```

Diese Methode erlaubt es dem Auto in einem kleinem Bogen nach links abzubiegen.

Das Auto fahrt nach vorne.

Die „turn_left_backward“ Methode

```
1  void turn_left_backward(int speed = 200) {
2    // Im bogen Links einbiegen
3
4    motor_left_backward();
5    motor_right_backward();
6
7    analogWrite(h_br_en1, speed);
8    analogWrite(h_br_en2, speed/4);
9  }
```

Diese Methode erlaubt es dem Auto in einem kleinem Bogen nach links abzubiegen.

Das Auto fahrt nach hinten.

Die „turn_right_backward“ Methode

```
1  void turn_right_backward(int speed = 200) {
2    // Im bogen Links einbiegen
3
4    motor_left_backward();
5    motor_right_backward();
6
7    analogWrite(h_br_en1, speed/4);
8    analogWrite(h_br_en2, speed);
9  }
```

Diese Methode erlaubt es dem Auto in einem kleinem Bogen nach rechts abzubiegen.

Das Auto fahrt nach hinten.

Die „turn_right_90_degrees“ Methode

```
1   void turn_right_90_degrees(int speed = 125) {
2     // Faehrt im Bogen 90 grad rueckwerts
3     stop();
4
5     digitalWrite(h_br_in1, LOW);
6     digitalWrite(h_br_in2, HIGH);
7     digitalWrite(h_br_in3, HIGH);
8     digitalWrite(h_br_in4, LOW);
9     analogWrite(h_br_en1, speed);
10     analogWrite(h_br_en2, 1);
11
12     delay(3000); // Adjust as needed
13
14     stop();
15   }
```

Diese Methode erlaubt es dem Auto in einem kleinem Bogen nach rechts abzubiegen.

Der Bogen ist größer als bei „turn_right_backward“.

Das Auto fahrt nach hinten.

Die “turn_left_90_degrees“ Methode

```
1
2   void turn_left_90_degrees(int speed = 125) {
3     // Faehrt im Bogen 90 grad rueckwerts
4     stop();
5     digitalWrite(h_br_in1, HIGH);
6     digitalWrite(h_br_in2, LOW);
7     digitalWrite(h_br_in3, LOW);
8     digitalWrite(h_br_in4, HIGH);
9     analogWrite(h_br_en1, speed);
10     analogWrite(h_br_en2, 1);
11
12     delay(3000); // Adjust as needed
13
14     stop();
15   }
```

Diese Methode erlaubt es dem Auto in einem kleinem Bogen nach links abzubiegen.

Der Bogen ist größer als bei „turn_left_backward“.

Das Auto fahrt nach hinten.

Die „spin_right“ Methode

```
1   void spin_right(int speed = 200){
2     // Auf der Stelle rechts drehen
3     motor_right_forward();
4     motor_left_backward();
5
6     analogWrite(h_br_en1, speed);
7     analogWrite(h_br_en2, speed);
8   }
```

Diese Methode erlaubt es dem Auto auf der Stelle nach rechts zu drehen, was bei feinen Bewegungen, wie bei der Linien Verfolgung, sehr Praktisch ist.

Die „spin_left“ Methode

```
1   void spin_left(int speed = 200){
2     // Auf der Stelle links drehen
3     motor_right_backward();
4     motor_left_forward();
5
6     analogWrite(h_br_en1, speed);
7     analogWrite(h_br_en2, speed);
8   }
```

Diese Methode erlaubt es dem Auto auf der Stelle nach links zu drehen, was bei feinen Bewegungen, wie bei der Linien Verfolgung, sehr Praktisch ist.

Die „motor_left_forward“ Methode

```
1  void motor_left_forward() {
2    //Motor links vorwaerts
3    digitalWrite(h_br_in1, HIGH);  // Motor 1 vor
4    digitalWrite(h_br_in2, LOW);
5  }
```

Dies ist eine Hilfemethode zum Einstellen der linken Motoren.

Die „motor_right_forward“ Methode

```
1  void motor_right_forward() {
2    //Motor rechts vorwaerts
3    digitalWrite(h_br_in3, LOW);  // Motor 1 vor
4    digitalWrite(h_br_in4, HIGH);
5  }
```

Dies ist eine Hilfemethode zum Einstellen der rechten Motoren.

Die „motor_left_backward“ Methode

```
1  void motor_left_backward() {
2    //Motor links rueckwaerts
3    digitalWrite(h_br_in1, LOW);  // Motor 1 rückwärts
4    digitalWrite(h_br_in2, HIGH);
5  }
```

Dies ist eine Hilfemethode zum Einstellen der linken Motoren.

Die „motor_right_backward“ Methode

```
1  void motor_right_backward() {
2    //Motor rechts rueckwerts
3    digitalWrite(h_br_in3, HIGH);  // Motor 1 rückwärts
4    digitalWrite(h_br_in4, LOW);
5  }
```

Dies ist eine Hilfemethode zum Einstellen der rechten Motoren.

Die „motor_right_stop“ Methode

```
1  void motor_right_stop() {
2    //Motor rechts stoppen
3    digitalWrite(h_br_in3, LOW);  // Motor 1 vor
4    digitalWrite(h_br_in4, LOW);
5  }
```

Diese Hilfemethode dient dazu, die rechten Motoren zu stoppen.

Die „motor_left_stop“ Methode

```
1  void motor_left_stop() {
2    // Motor links stoppen
3    digitalWrite(h_br_in1, LOW);  // Motor 1 vor
4    digitalWrite(h_br_in2, LOW);
5  }
```

Diese Hilfemethode dient dazu, die linken Motoren zu stoppen.

Die LineTracking Klasse

Attribute

```
1  private:
2    Hbridge h_bridge; // Benoetig Zugriff auf die H-Bruecke zum steuern der Motoren
```

Methoden

Der Konstruktor

```
1    explicit LineTracking(Hbridge h_bridge){
2      this->h_bridge = h_bridge;
3    }
```

Die Klasse benötigt Zugriff auf die H-Brücke zum Steuern der Motoren, daher wird ein Instanziiertes H-Brücken Objekt zum erstellen eines Objektes dieser Klasse benötigt.

Die „get_sensor_vals“ Methode

```
1   void get_sensor_vals(){
2     //Die Aktuellen Sensorwerte einlesen
3     // Links, Mitte, Rechts
4     ls_links = !digitalRead(lt_links);
5     ls_mitte = !digitalRead(lt_mitte);
6     ls_rechts = !digitalRead(lt_rechts);
7
8     return;
9   }
```

Diese Methode liest die aktuellen werde der Sensoren ein.

Die „line_tracking“ Methode

```
1    void line_tracking(){
2      //Linetracking Funktion
3      get_sensor_vals();
4      if (ls_mitte) {
5          h_bridge.drive_forward(150);
6      } else if (ls_rechts) {
7          h_bridge.spin_right();
8
9          while (ls_rechts) {
10           h_bridge.spin_right();
11           get_sensor_vals();
12         }
13         h_bridge.stop();
14     } else if (ls_links) {
15         h_bridge.spin_left();
16
17         while (ls_links) {
18           get_sensor_vals();
19           h_bridge.spin_left();
20         }
21         h_bridge.stop();
22     } else {
23
24         h_bridge.stop();
25     }
26   }
```

Diese Methode beinhaltet die Logik für die Linienverfolgung.

Die Car Klasse

Attribute

```
1  class Car { // Klasse zum Steuern des Autos zum vereinfachen der Steuerung
2    public:
3      String mode{}; // Modus wie z.b 'manuell'
4      ServoMotor servo_motor = ServoMotor(); //Servo Motor Initialisieren
5      Hbridge h_bruecke = Hbridge(); // H-Bruecke Initialisieren
6      LineTracking line_tracking_modul = LineTracking(h_bruecke); // LineTracking Modul
Initialisieren
7      bool on_off = false; // Auto An/Aus Schalten
8      int** data_array; // Platzhalter fuer die get_cords() Funktion aus dem Servo Motor
9      String richtung{}; //WIRD FUER DEN MANUELLEN MODUS VERWENDET
10
```

Methoden

Der Konstruktor

```
1      explicit Car(Servo new_servo) {
2        // Konstruktor der Klasse
3        mode = "manuell";
4        this->servo_motor.set_servo(new_servo);
5        this->richtung = "STOP";
6      }
```

Die Car Klasse benötigt einen Instanziiertes Servo Objekt, um erstellt zu werden.

Dies wurde getan, da es nicht möglich war, das Servo Objekt innerhalb des Konstruktors zu erstellen.

Die „infra_red_handler“ Methode

```
1  void infra_red_handler(){
2      // Funktion zur Kontrolle via Infrarot
3      if (IrReceiver.decode())// Wenn etwas im Infrarot buffer steht
4       {
5        IR_VALUE value = (IR_VALUE) IrReceiver.decodedIRData.command;
6        IrReceiver.resume();
7        switch (value) { // Kontroll Logik
8          //Modus wechsel => # + 1/2/3
9          case IR_VALUE::HASHTAG:
10           while (!IrReceiver.decode()) {
11             continue;
12           }
13           switch ((IR_VALUE) IrReceiver.decodedIRData.command){
14             case IR_VALUE::VAL_1:
15             this->mode = "manuell";
16             break;
17           case IR_VALUE::VAL_2:
18             this->mode = "automatik";
19             break;
20           case IR_VALUE::VAL_3:
21             this->mode = "line_tracking";
22             break;
23           default:
24             break;
25           }
26         case IR_VALUE::VOR:
27           this->richtung = "VOR";
28           break;
29         case IR_VALUE::ZURRUECK:
30           this->richtung = "ZURRUECK";
31           break;
32         case IR_VALUE::RECHTS:
33           this->richtung = "RECHTS";
34           break;
35         case IR_VALUE::LINKS:
36           this->richtung = "LINKS";
37           break;
38         case IR_VALUE::OK:
39           this->richtung = "STOP";
40         default:
41           break;
42       }
43    }
44  }}
```

Diese Methode kontrolliert nach jedem Ausführen der 3 Hauptfunktionen, ob etwas im Infrarot Buffer steht und wertet die Ergebnisse aus.

Dies kann ein Wesel von des Modus sein oder eine Änderung der Richtung.

Diese Funktion ist im aktuellen Programm nicht angebunden, da wir Bluetooth verwenden.

Werte Tabelle

Kombination	Bedeutung
^ → 70 (Ganzzahl) → Pfeil Hoch	Vor Fahren
> → 67 (Ganzzahl) → Pfeil Rechts	Nach Rechts fahren
< → 68 (Ganzzahl) → Pfeil Links	Nach Links Fahren
∨ → 21 (Ganzzahl) → Pfeil Runter	Zurück Fahren
OK → 64 (Ganzzahl) → OK in der Mitte	Stoppen
# + 1	Modus in „Manuell“ wechseln
# + 2	Modus in „Automatik“ wechseln
# + 3	Modus in „Line Tracking“ wechseln

Die Werte Tabelle wird in Form der Enum Klassen implementiert.

Die „bluetooth_handler“ Methode

```
1   void bluetooth_handler(){
2     // Funktion zum Handlen von Bluetooth Modul
3     if(Serial.available()){ // Wenn Etwas im Buffer steht
4       BT_VALUE value = (BT_VALUE) Serial.read();
5       switch (value) { // Kontroll Logik
6         case BT_VALUE::VOR:
7           this->richtung = "VOR";
8           break;
9         case BT_VALUE::VORLINKS:
10          this->richtung = "VORLINKS";
11          break;
12        case BT_VALUE::VORRECHTS:
13          this->richtung = "VORRECHTS";
14          break;
15        case BT_VALUE::ZURRUECK:
16          this->richtung = "ZURRUECK";
17            break;
18        case BT_VALUE::LINKS:
19          this->richtung = "LINKS";
20          break;
21        case BT_VALUE::ZURRUECKLINKS:
22          this->richtung = "ZURRUECKLINKS";
23          break;
24        case BT_VALUE::ZURRUECKRECHTS:
25          this->richtung = "ZURRUECKRECHTS";
26          break;
27        case BT_VALUE::RECHTS:
28          this->richtung = "RECHTS";
29          break;
30        case BT_VALUE::STOP:
31          this->richtung = "STOP";
32          break;
33        case BT_VALUE::MANUELL:
34          this->mode = "manuell";
35          break;
36        case BT_VALUE::AUTOMATIK:
37          this->mode = "automatik";
38          break;
39        case BT_VALUE::TRACKING:
40          this->mode = "line_tracking";
41          break;
42        default:
43          break;
44      }
45    }
46  }
```

Diese Methode kontrolliert nach jedem ausführen der 3 Hauptfunktionen, ob etwas im Seriellen Buffer steht und wertet die Ergebnisse aus.

Dies kann ein Wesel von des Modus sein oder eine Änderung der Richtung.

Werte Tabelle

Command/Wert	Bedeutung
„f“	Vor fahren
„q“	Vorne Links Fahren
„e“	Vorne Rechts Fahren
„z“	Zurück Fahren
„x“	Zurück Links Fahren
„c“	Zurück Rechts Fahren
„r“	Rechts Fahren
„l“	Links Fahren
„s“	Stop
„m“	Manuell Modus
„t“	Tracking Modus
„a“	Automatik Modus

Die Werte Tabelle wird in Form der Enum Klassen implementiert.

```
1   void print_cords(){
2     // Funtion zum Printen der Aktuellen Koordinaten im Umkreis
3     int** data_array = servo_motor.get_cords();
4     for (int i = 0; i < 13; ++i) {
5       Serial.print("[");
6       Serial.print(i);
7       Serial.print("] Degrees: ");
8       Serial.print(data_array[i][0]);
9       Serial.print(" Distance: ");
10       Serial.print(data_array[i][1]);
11       Serial.println();
12    }
13     deconstruct_data_array(data_array);
14     servo_motor.deconstruct_data_array();
15    }
```

Gibt die Koordinaten rund um das Auto im seriellen Monitor aus.

Die „deconstruct_data_array“ Methode

```
1    void deconstruct_data_array(int** data_array){
2      for (int i = 0; i < 13; ++i) {
3        delete[] data_array[i];
4    }
5        delete[] data_array;
6    }
```

Diese Methode gibt den Speicher nach aufrufen der get_cords() Funktion wieder frei.

Wenn dies nicht gemacht wird geht dem Auto nach einer weile der RAM aus und das Programm stürzt ab!

Die „manuell“ Methode

```
 1   void manuell(){
 2     //Funktion fuer den Manuellen Modus
 3     if(this->richtung.equals("VOR")){
 4       h_bruecke.drive_forward(255);
 5     }else if (this->richtung.equals("VORRECHTS")) {
 6       h_bruecke.turn_right(255);
 7     }else if (this->richtung.equals("VORLINKS")) {
 8       h_bruecke.turn_left(255);
 9     }else if (this->richtung.equals("ZURRUECK")) {
10        h_bruecke.drive_backward(255);
11      }else if (this->richtung.equals("ZURRUECKLINKS")) {
12        h_bruecke.turn_left_backward(255);
13      }else if (this->richtung.equals("ZURRUECKRECHTS")) {
14        h_bruecke.turn_right_backward(255);
15      }
16
17      else if (this->richtung.equals("RECHTS")) {
18        h_bruecke.spin_right(255);
19      }else if (this->richtung.equals("LINKS")) {
20        h_bruecke.spin_left(255
21        );
22      }else {
23        h_bruecke.stop();
24      }
25    }
```

Diese Methode implementiert die Logik für den manuellen Modus.

Die „automatisches_fahren_mit_ausweichen“ Methode

```
 1   void automatisches_fahren_mit_ausweichen(int sicherheitsabstand = 20) {
 2     //Funktion fuer den Automatik modus
 3     int entfernung = this->servo_motor.get_frontal_distance();
 4     if (entfernung < sicherheitsabstand) {
 5         h_bruecke.drive_backward();
 6         delay(2000); // 2 Sekunden warten
 7         h_bruecke.stop(); // Anhalten
 8         h_bruecke.turn_right_90_degrees(); // Nach rechts abbiegen
 9         delay(1000); // 1 Sekunde warten
10          h_bruecke.stop(); // Anhalten
11          h_bruecke.drive_forward(); // Vorwärts fahren
12        } else{
13          h_bruecke.drive_forward();
14        }
15  }
```

Diese Methode implementiert die Logik für das automatische Fahren des Autos.

Die „start“ Methode

```
1    void start(){ // Funktion zum Starten des Autos bzw. unser Main Loop
2      this->on_off = true;
3      while(on_off){
4        //Verschiedene Modis Starten
5        while(this->mode.equals("line_tracking")){
6          //Linetracking starten
7          line_tracking_modul.line_tracking();
8          // Schauen, ob es veraenderungen im Bluetooth Modul gibt, wie z.b neue Daten im
Buffer
9          bluetooth_handler();
10        }
11        while (this->mode.equals("manuell")) {
12          //Manuell starten
13          manuell();
14          // Schauen, ob es veraenderungen im Bluetooth Modul gibt, wie z.b neue Daten im
Buffer
15          bluetooth_handler();
16        }
17        while (this->mode.equals("automatik")) {
18          //Automatik starten
19          automatisches_fahren_mit_ausweichen();
20          // Schauen, ob es veraenderungen im Bluetooth Modul gibt, wie z.b neue Daten im
Buffer
21          bluetooth_handler();
22        }
23
24      }
25    }
```

Die „Start“ Methode ist das Herzstück des Arduino Car Projektes. Es verbindet die einzelnen Module des Autos und kümmert sich um die Verknüpfung der Logik.

Die bluetooth_handler() Methode bedeutet, dass aktuell der Bluetooth Modus an ist und kann durch die infra_red_handler() Methode getauscht werden, um Infra rot einzuschalten.

Quellen

Bluetooth Basics: https://arduinogetstarted.com/tutorials/arduino-bluetooth

Infrarot Basics: https://www.makerguides.com/ir-receiver-remote-arduino-tutorial/

Line Tracking Basics: https://www.electroniclinic.com/ky-033-line-tracking-sensor-arduino-circuit-and-programming/

H-Bruecke Basics: https://funduino.de/nr-34-motoren-mit-h-bruecke-l298n-ansteuern

Servo Motor Basics: https://funduino.de/nr-12-servo-ansteuern

Ultraschall Basics: https://arduinogetstarted.com/tutorials/arduino-ultrasonic-sensor

Arduino IDE 2.0: https://docs.arduino.cc/software/ide-v2/tutorials/getting-started/ide-v2-downloading-and-installing/

Github Repo mit Code: https://github.com/Kai7788/ArduinoCar

Commits von Kai: https://github.com/Kai7788/ArduinoCar/commits?author=Kai7788

Commits von Jonas: https://github.com/Kai7788/ArduinoCar/commits?author=Jonasbho

App zum Steuern: https://play.google.com/store/apps/details?id=air.com.elegoo.elegooTool&hl=en_US

Erstellung der Diagramme mit Draw.io: https://app.diagrams.net/

Dokumentation Geschrieben mit LibreOffice Writer: https://de.libreoffice.org/download/download/

Erstellen der Code-Blöcke: https://extensions.libreoffice.org/en/extensions/show/5814