HTWK

Hochschule für Technik,
Wirtschaft und Kultur Leipzig

Fakultät Ingenieurwissenschaften

Modul E469 - Ausgewählte Themen der Automatisierungs

Dobots AI-Starter

Author Michael Bäumler Matrikel-Nr.: 80501
Aymane Elhamdouni Matrikel-Nr.: 78669
Betreuer Prof. Dr.-Ing. Mirco Fuchs
Prof. Dr.-Ing. Jens Jaekel
M.Sc. Mathias Mueller

17. September 2024

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

1 Einführung

Dieses Projekt wurde im Rahmen des Moduls **Äusgewählte Themen der Automatisierungstechnik"** durchgeführt. Unser Ziel war es, den **Dobot AI-Starter** so zu programmieren, dass er Linien auf dem Boden verfolgt, Farben erkennt und mithilfe von Ultraschallsensoren Hindernisse erkennt.

Zunächst nutzten wir die von Dobot bereitgestellten Codes aus der offiziellen Guideline, um diese Funktionen zu implementieren. Da diese jedoch nicht wie erwartet funktionierten, entwickelten wir eigene Codes zur Lösung der Probleme.

Zielsetzung

Das Ziel dieses Projekts ist es, ein funktionsfähiges System zu entwickeln, das:

1. Linien auf dem Boden selbstständig erkennt und diesen folgt (Linienverfolgung).
2. Verschiedene Farben identifiziert und entsprechend darauf reagiert (Farberkennung).
3. Hindernisse mittels eines Ultraschallsensors erkennt und daraufhin seine Bewegung anpasst (Abstandsmessung).

1.1 Aufgabenstellung:

- **Entwicklung und Implementierung eines Linienverfolgungssystems:** Das Fahrzeug muss in der Lage sein, eine vorgegebene schwarze Linie auf einem weißen Untergrund selbstständig zu erkennen und zu verfolgen.
- **Integration der Farberkennung:** Das System soll in der Lage sein, Farben auf der Linie zu erkennen und darauf zu reagieren.
- **Einsatz von Ultraschallsensoren zur Hinderniserkennung:** Der Ultraschallsensor soll Hindernisse in der Fahrbahn des Fahrzeugs detektieren. Bei Hinderniserkennung muss das Fahrzeug stoppen und eine alternative Route wählen.

Alle technischen Details, die vollständigen Codes sowie eine Präsentation der funktionierenden Systeme sind auf GitHub hier zu finden. Die Dokumentation ist so mit der Idee erstellt worden, dass Studierende in der Lage sein sollen, die Programme nachzuvollziehen und eigene Anpassungen vornehmen zu können.

Überblick Projekt

1. **Dobot AI-Starter**
2. **Linien- und Farberkennungs Programm**
3. **Objektvermeidungs-Programm**
4. **Analyse der Dobot-Beispiele**
5. **Installation und Setup**
6. **Ausblick und Verbesserungpotential**

2 Übersicht: AI-Starter

Abbildung 1: Der AI-Starter Roboter

Der **AI-Starter** ist ein Lernroboter, der auf der **Arduino Mega2560**-Plattform basiert und speziell für Bildungszwecke von **Dobot** entwickelt wurde. Er kombiniert verschiedene Sensoren und Aktoren, die es ermöglichen, Aufgaben wie **Hindernisvermeidung**, **Linienverfolgung** und **Farberkennung** auszuführen. Das modulare Design und die einfache Programmierbarkeit machen ihn zu einem idealen Werkzeug für den Einstieg in die Robotik und Programmierung. Der AI-Starter besteht aus den folgenden Komponenten/ technischen Daten:

Hauptkomponente	**Beschreibung**
Steuerplatine	DuDuino Mega (kompatibel mit Arduino Mega 2560)
MPU	ATmega2560
Batterie	18650 Li-Ionen-Akku wiederaufladbar
Ultraschallsensoren	3 Ultraschallsensoren zur Distanzmessung
Farbsensoren	2 Farbsensoren zur Erkennung von Rot und Grün
Infrarot-Linienverfolgungsmodul	1 Modul zur Erkennung von Linien
Geomagnetischer Sensor	1 geomagnetischer Sensor für Richtungsbestimmung
Lichtsensor	1 Lichtsensor

Tabelle 1: Hauptkomponenten des AI-Starters

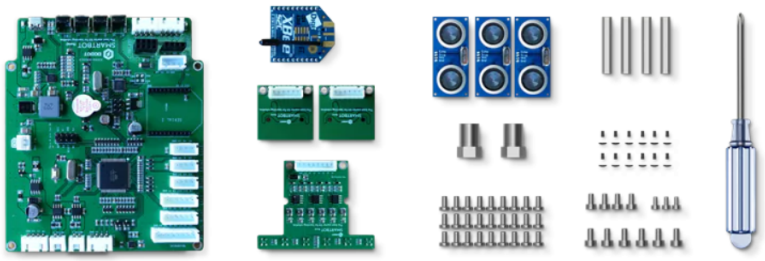

Abbildung 2: Bauteile des AI-Starters

Technische Daten	**Beschreibung**
Betriebsspannung	7,4V
Größe	195mm × 172mm × 79mm
Gewicht	810g
Maximale Last	500g
Reifendurchmesser	67mm
Betriebstemperatur	0°C 40°C
Steuerungssoftware	Arduino IDE, Mixly
Kommunikationsschnittstelle	USB, seriell
Erweiterungsschnittstelle	4PIN I/O Schnittstelle *2
Motorparameter	Untersetzungsverhältnis: 48 : 1, Spannung: 7V, Leerlaufstrom: 150mA, Blockierstrom: 700mA, Drehzahl: 100 U/min, Encoder: 585 Pulse/Umdrehung

Tabelle 2: Technische Daten des AI-Starters

Da die von Dobot bereitgestellten *example Codes* nicht wie erwartet funktionierten, haben wir eigene Programmentwürfe für die **Linien- und Farberkenung** sowie für die **Objektvermeidung** erstellt, die in den folgenden Kapiteln vorgestellt werden. Der Analyse der nicht funktionierten Codes wird sich in Kapitel 5 gewidmet.

3 Linien- und Farberkennungs Programm

3.1 PID Regler

Ein effektives Linienverfolgungssystem erfordert, dass die Räder separat gesteuert werden, um Fehler zu korrigieren und Abweichungen zu minimieren. Hier kommt ein **PID-Controller** (Proportional-Integral-Derivative) ins Spiel. Ein PID-Controller vergleicht die aktuelle Position des Fahrzeugs mit der Soll-Position (der Linie) und berechnet basierend auf der Differenz (dem Fehler) die notwendigen Korrekturen.

- Der *Proportional*-Anteil reagiert auf den aktuellen Fehler, indem er proportional zur Abweichung die Radgeschwindigkeit anpasst.
- Der *Integral*-Anteil summiert vergangene Fehler, um systematische Abweichungen zu korrigieren,
- und der *Derivative*-Anteil versucht, zukünftige Fehler vorherzusagen, indem er die Änderungsrate des Fehlers berücksichtigt.

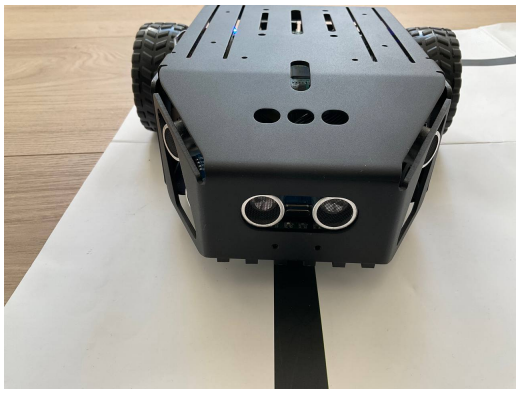

Abbildung 3: Die Mittleren Sensoren lesen die Schwarze Linie → Keine Fehler

Abbildung 4: Die Linken Sensoren lesen die Schwarze Linie → Fehler.

Durch die unabhängige Steuerung der Räder kann der PID-Controller den Unterschied in der Geschwindigkeit zwischen dem linken und rechten Rad anpassen, um das Fahrzeug auf der Linie zu halten. Wenn beispielsweise das Fahrzeug nach links von der Linie abweicht, wird die Geschwindigkeit des linken Rades erhöht und die des rechten Rades verringert, um das Fahrzeug wieder auf den Kurs zu bringen. Ohne diese getrennte Kontrolle der Räder ist eine präzise Linienverfolgung nicht möglich, da beide Räder in die gleiche Richtung und mit der gleichen Geschwindigkeit fahren.

3.2 Entwürfe zur Linien- und Farberkennung

Zuerst beschäftigen wir uns mit der Linien und Farberkennung. Dafür haben wir nach ausgiebiger Recherche, unter anderem inspiriert durch das Arduino-Projekt „Line Following Robot“ (siehe hier) ein Programm entwickelt, dass auf den Funktionen der von Dobot bereitgestellten Bibliothek aufbaut. Dieses Programm kombiniert die Erkennung von Linien mittels Infrarotsensoren mit der Detektion von Farben, um eine Navigation zu ermöglichen, ohne zusätzlich einen PID-Reglers programmieren zu müssen.

Linienverfolgung:

Das Programm liest die Werte von sechs Infrarotsensoren (IR), die an den Unterboden des AI-Starters angebracht sind, um die auf der Testmatte vorgegebenen schwarzen Linien zu verfolgen. Die Daten der IR-Sensoren werden bitweise in einer Variablen zusammengeführt, die den Zustand der Linie (ob erkannt oder nicht) für jeden Sensor speichert.

```
    int getIRState() {
    int irstate = 0;
    for (int i = 0; i < 6; i++) {
        irstate |= AIStarter_SmartBotGetIRModuleValue(i) << i;
    }
    return irstate;
}
```

Basierend auf dieser Information steuert das Programm die Bewegungen des AI-Starters, um die Linie präzise zu folgen. Unterschiedliche Bitmuster werden analysiert, um zu bestimmen, ob der Roboter nach links oder rechts ausweichen oder weiter geradeaus fahren muss.

Der folgende Codeblock zeigt die Hauptlogik der Linienverfolgung:

```
\\ 0 : Keine Schwarze Linie
\\ 1 : Schwarze Linie
\\ int Speed = 150;

if (irstate == 0b001100) {   \\Die Mittleren Sensor (IR3 && IR4) lesen die Schwarze Linie
    AIStarter_SmartBotSetMovment(FRONT, speed);
   \\geradeaus fahren
} else if (irstate == 0b000110) {   \\ Die rechten Sensor (IR4 und IR5) lesen die Schwarze
     Linie
    AIStarter_SmartBotSetMovment(RIGHT, speed);
  \\rechts fahren
}
```

Hierbei handelt es sich um eine einfache Bedingung, bei der das IR-Modul erkennt, ob sich der Roboter exakt auf der Linie befindet (Muster: `0b001100`), und ihn entsprechend steuert. Je nach Position wird die Richtung des Roboters angepasst.

Vielleicht hier noch ein foto von der LineTracking Matte mit dem Ai-starter drauf von oben?

Farberkennung:

Zur Farberkennung verwendet der AI-Starter zwei Farbsensoren. Diese sind in der Lage, die Intensität der Rot-, Grün- und Blauanteile des erfassten Lichts zu messen. Durch Vergleich der Farbsensorwerte mit definierten Schwellenwerten kann das Programm erkennen, ob eine rote oder grüne Farbe vorliegt. Wird eine Farbe erkannt, gibt der Roboter akustische Signale aus (z.B. zweimaliges Piepen bei Rot, dreimaliges Piepen bei Grün), um den Benutzer über die erkannte Farbe zu informieren.

Der folgende Code überprüft, ob die Farbe Rot erkannt wird:

```
int Threshold = 20;
redValue1: Der Farben Sensor 1 liest farbe "Rot"
redValue2: Der Farben Sensor 2 liest farbe "Rot"

```

```
 greenValue1: Der Farben Sensor 1 liest farbe "Gr n"
 greenValue2: Der Farben Sensor 2 liest farbe "Gr n"

blueValue1: Der Farben Sensor 1 liest farbe "Blau"
blueValue2: Der Farben Sensor 2 liest farbe "Blau"

 if ((redValue1 - greenValue1 > Threshold && redValue1 - blueValue1 > Threshold) &&
     (redValue2 - greenValue2 > Threshold && redValue2 - blueValue2 > Threshold)) {
    if (colorState != RED) {
        colorState = RED;
        Serial.println("Rot erkannt - Beep");
        // Tiefes Piepen als Signal
    }
}
```

Hierbei werden die Rot-, Grün- und Blauwerte der beiden Sensoren miteinander verglichen, um eine zuverlässige Farberkennung zu gewährleisten. Die Nutzung zweier Sensoren sorgt für eine erhöhte Genauigkeit und Stabilität bei der Farbdetektion.

4 Objektvermeidungs-Programm

4.1 Entwürfe für die Objektvermeidung

Nach der erfolgreichen Implementierung der Linien- und Farberkennung wandten wir uns der Entwicklung eines Programms zur Hindernisvermeidung für den AI-Starter zu. Hierbei nutzen wir die Ultraschallsensoren des Roboters, um Hindernisse zu erkennen und angemessen darauf zu reagieren.

Objektvermeidung: Mithilfe von drei Ultraschallsensoren (links, rechts und vorne) werden die Abstände zu potenziellen Hindernissen gemessen. Überschreitet der gemessene Abstand einen vordefinierten Schwellenwert (`DIS`), entscheidet das Programm automatisch, ob der Roboter nach links, rechts oder zurückfahren muss. Falls keine Hindernisse erkannt werden, setzt der Roboter seine Fahrt nach vorne fort. Die Funktion `avoidObstacles()` bildet das Herzstück dieser Entscheidungslogik.

Der folgende Codeblock zeigt die Logik der Hindernisvermeidung:

```
#define DIS 10
void avoidObstacles() {
    float distanceLeft = AIStarter_SmartBotGetSonar(SONAR3);
    float distanceRight = AIStarter_SmartBotGetSonar(SONAR1);
    float distanceFront = AIStarter_SmartBotGetSonar(SONAR2);

    if (distanceFront < DIS) { // der vordere Sensor lest ein Abstand mehr als DIS
        AIStarter_SmartBotSetMovment(BACK, 100);
        delay(1000);
    } else if (distanceLeft < DIS) { // der linke Sensor lest ein Abstand mehr als DIS
        AIStarter_SmartBotSetMovment(RIGHT, 100);
        delay(1000);
    } else if (distanceRight < DIS) { // der rechte Sensor lest ein Abstand mehr als DIS
        AIStarter_SmartBotSetMovment(LEFT, 100);
        delay(1000);
    } else {
        AIStarter_SmartBotSetMovment(FRONT, 100);
    }
}
```

Durch diese Bedingungen stellt das Programm sicher, dass der Roboter in Situationen mit nahegelegenen Hindernissen flexibel reagiert. Es wird stets überprüft, ob links oder rechts mehr Platz zum Ausweichen vorhanden ist, und daraufhin die Fahrtrichtung angepasst. Falls keine Hindernisse vorliegen, setzt der Roboter seine Vorwärtsbewegung fort.

5 Analyse der Dobot-Beispiele

Im Folgenden analysieren wir die Motorsteuerung und die auftretenden Probleme bei der Implementierung.

5.1 Ursprünglicher Code zur Funktion der Motorsteuerung

Die Funktion `SET_MOTOR_SPEED()` steuert die Anpassung der Geschwindigkeit der beiden Motoren, `MOTORR` und `MOTORL`. Diese Funktion soll die Geschwindigkeit schrittweise erhöhen oder verringern, um abrupte Bewegungen zu verhindern.

Zu Beginn prüft die Funktion, ob die Zielgeschwindigkeit der Motoren bereits mit der letzten Zielgeschwindigkeit übereinstimmt.

- Wenn dies der Fall ist, wird keine Änderung vorgenommen. Andernfalls wird der aktuelle Zustand der Motoren gespeichert und eine lineare Beschleunigung durchgeführt.
- Dieser Prozess erfolgt über eine Schleife, in der die Geschwindigkeit in zehn Schritten angepasst wird, um eine sanfte Beschleunigung oder Verzögerung zu simulieren.
- Am Ende der Schleife werden die neuen Zielgeschwindigkeiten gespeichert, um sie für den nächsten Vergleich bereitzuhalten.

Der folgende Code zeigt diese Logik:

```
void SET_MOTOR_SPEED(int MOTORR_tar_speed, int MOTORL_tar_speed)
{
    static int num = 0;
    static int MOTORL_last_tar_speed = 0;
    static int MOTORL_last_speed = 0;
    static int MOTORL_now_speed = 0;

    static int MOTORR_last_tar_speed = 0;
    static int MOTORR_last_speed = 0;
    static int MOTORR_now_speed = 0;

    if(MOTORR_tar_speed == MOTORR_last_tar_speed && MOTORL_tar_speed == MOTORL_last_tar_speed)
    {
        // Keine Anpassung notwendig
    }
    else {
        num = 0;
        MOTORL_last_speed = MOTORL_now_speed;
        MOTORR_last_speed = MOTORR_now_speed;
        while(num <= 10) {
            num++;
            MOTORL_now_speed += (MOTORL_tar_speed - MOTORL_last_speed) / 10;
            MOTORR_now_speed += (MOTORR_tar_speed - MOTORR_last_speed) / 10;
            AIStarter_SmartBotSetMotor(MOTORR, MOTORR_now_speed);
            AIStarter_SmartBotSetMotor(MOTORL, MOTORL_now_speed);
            delay(50);
        }
        MOTORR_last_tar_speed = MOTORR_tar_speed;
        MOTORL_last_tar_speed = MOTORL_tar_speed;
    }
}
```

Trotz der theoretischen Korrektheit dieser Funktion traten bei unseren Tests Probleme auf, die sich vor allem in der tatsächlichen Umsetzung der Motorgeschwindigkeit zeigten. Diese Probleme sind eng mit der Funktion `AIStarter_SmartBotSetMotor()` verbunden, die in der Praxis nicht wie erwartet funktioniert hat.

5.2 Probleme mit der Funktion `AIStarter_SmartBotSetMotor`

Die Funktion `AIStarter_SmartBotSetMotor(int port, int speed)` wird in den Beispielcodes von Dobot, wie `LineControl`, `ObstacleAvoid` und `ColorRecognition`, verwendet. Der serielle Monitor zeigte zwar die korrekten Werte für die Sensoren und die berechneten Geschwindigkeiten der Räder an, dennoch bewegte sich das Fahrzeug nicht wie vorgesehen.

Irstate	**speedLeftWheel [rpm]**	**speedRightWheel [rpm]**
63 = 0b111111	50	50
55 = 0b001100	47	53
54 = 0b110000	78	22
53 = 0b000011	22	78
52 = 0b100000	95	05
51 = 0b000001	05	95

Tabelle 3: Serial Monitor Ausgänge

Der Zweck dieser Funktion besteht darin, die Geschwindigkeit der Motoren individuell anzupassen. Dabei sollte es möglich sein, unterschiedliche Geschwindigkeiten für `MOTORR` und `MOTORL` festzulegen, um eine präzise Steuerung des Roboters zu ermöglichen. In der Praxis zeigte sich jedoch, dass die Fahrzeuge immer mit einer konstanten Geschwindigkeit fuhren, unabhängig von den übergebenen Werten.

Dieses Verhalten deutet darauf hin, dass die interne Implementierung der Funktion entweder fehlerhaft ist oder dass die Motorsteuerung des AI-Starters nicht korrekt mit den übergebenen Geschwindigkeiten arbeitet. Da dies in mehreren Beispielcodes vorkam, haben wir alternative Ansätze zur Geschwindigkeitssteuerung entwickeln, um dieses Problem zu umgehen.

5.3 Analyse der Funktion `getCurrentPos`

Die Funktion `getCurrentPos()` berechnet die aktuelle Position des Fahrzeugs relativ zur Linie, indem sie die Daten der sechs Infrarotsensoren (IR-Sensoren) auswertet.

Der Algorithmus summiert die Positionen der aktiven Sensoren, also der Sensoren, die eine Linie detektieren, und berechnet daraus die Abweichung des Fahrzeugs von der Mittellinie. Ein wichtiger Aspekt der Funktion ist die Verwendung eines Filters, um abrupte Änderungen zu vermeiden. Wenn kein Sensor eine Linie erkennt, wird die letzte bekannte Position des Fahrzeugs beibehalten, um ein plötzliches Abdriften zu verhindern. Dieser gleitende Durchschnitt sollte dazu beitragen, eine gleichmäßige Bewegung zu gewährleisten.

Die Funktion liefert einen Wert zurück, der die Position des Fahrzeugs relativ zur Linie darstellt und von der Steuerlogik verwendet wird, um die Richtung und Geschwindigkeit anzupassen.

```
float getCurrentPos(const int irstate) {
    const float coeff = 0.7;
    const int irPos[] = {-30, -18, -6, 6, 18, 30};
    static float lastPos;
    float curPos;
    float readPos;
    int total = 0;
    int irOffCnt = 0;

    for (int i = 0; i < IR_NUM; i++) {
        if (irstate & (1 << i)) {
            total += irPos[i];
            irOffCnt++;
        }
    }

    readPos = irOffCnt ? total / irOffCnt : lastPos;
    curPos = (1 - coeff) * lastPos + coeff * readPos;
    lastPos = curPos;

    return curPos;
}
```

5.4 Analyse der Funktion `setCarSpeed`

Die Funktion `setCarSpeed()` passt die Geschwindigkeit der Räder des Fahrzeugs auf Basis der Abweichung von der Linie an. Diese Anpassung erfolgt mithilfe eines PID-Reglers, der für eine präzise Steuerung sorgt. Der Regler berechnet einen Korrekturwert, der die Geschwindigkeiten der Räder so beeinflusst, dass das Fahrzeug der Linie stabil folgen kann. Der PID-Regler trägt dazu bei, dass der Roboter sowohl kleine als auch größere Abweichungen effektiv ausgleicht und die Bewegung gleichmäßig bleibt.

```
void setCarSpeed(const float curPos) {
    const int baseSpeed = 50;
    const float kp = 1;
    const float ki = 0.06;
    const float kd = 0.0;
    const float errorsumLimit = 50;
    float error = curPos;
    static float lastError;
    static float errorsum;
    float errorChange;
    int speedLeftWheel;
    int speedRightWheel;
    int speedOffset;

    errorsum += error;
    if (errorsum > errorsumLimit) errorsum = errorsumLimit;
    else if (errorsum < -errorsumLimit) errorsum = -errorsumLimit;

    errorChange = error - lastError;
    speedOffset = kp * error + ki * errorsum + kd * errorChange;
    lastError = error;

    speedLeftWheel = baseSpeed + speedOffset;
    speedRightWheel = baseSpeed - speedOffset;

    AIStarter_SmartBotSetMotor(MOTORL, speedLeftWheel);
    AIStarter_SmartBotSetMotor(MOTORR, speedRightWheel);
}
```

Der PID-Regler sorgt für eine dynamische Anpassung der Geschwindigkeit, indem er den Fehler, die Fehlerhistorie (integraler Anteil) und die Änderungsrate des Fehlers (differentieller Anteil) berücksichtigt.

6 Installation und Setup

Hardwareanschlüsse

Im GitHub-Repository Dobot AI-Starter ist eine detallierte Anleitung zum Aufbau der Hardware zu finden.

Benötigte Software

Für die Programmierung des Dobot AI-Starters wird die **Arduino IDE** verwendet. In diesem Projekt wurde die Version 1.8.19 eingesetzt, welche die Programmiersprache Arduino C unterstützt.

Um die Bibliothek für den AI-Starter zu installieren und das System korrekt zu konfigurieren, sind die folgenden Schritte zu befolgen:

- Installiere und öffne **Arduino IDE** und navigiere zu *Werkzeuge* → *Bibliotheken verwalten.*
- Suche im Bibliotheksverwalter über die Suchleiste nach *AIStarter* und installiere die entsprechende Bibliothek.
- Schließe den **AI-Starter** an den PC an.
- Gehe erneut zu *Werkzeuge* → *Board* und wähle *Arduino Mega or Mega2560* aus.
- Wähle im Menü *Werkzeuge* → *Prozessor* ebenfalls *Arduino Mega or Mega2560* aus.
- Gehe zu *Werkzeuge* → *Port* und wähle den entsprechenden COM-Port für *Arduino Mega or Mega2560.*
- Öffne den *seriellen Monitor* über *Werkzeuge* → *Serieller Monitor*, um Sensordaten auszulesen. Dies funktioniert nur, wenn der AI-Starter mit der Arduino IDE verbunden ist.

Code Implementierung

Im GitHub-Repository Dobot AI-Starter sind die von uns beschriebenen Programme und die Beispielcodes zu finden.

- Kopiere den Code in die **Arduino IDE**.
- Klicke oben links auf das Symbol mit dem **grünen Haken**, um den Code zu **kompilieren** bzw. **überprüfen**.
- Wenn der Code erfolgreich kompiliert wurde, klicke auf das Symbol mit dem **grünen Pfeil**, um das Programm auf den **AI-Starter hochzuladen**.

Testumgebung

Jetzt ist der AI-Starter fast einsatzbereit. Um eine geeignete Testumgebung herzustellen wird die von Dobot bereitgestellte Umgebungsmatte verwendet. Dabei handelt es sich um ein Plakat auf dem die Linien und Farbmarkierungen für den Dobot aufgedruckt sind. Um das Objektvermeidungs-Programm zu testen, wird empfohlen, Wände um die Testmatte zu platzieren, um geeignete Hindernisse zu simulieren. Dabei sollten die Wände so gestaltet sein, dass sie von den Sensoren des Dobot zuverlässig erkannt werden.

7 Ausblick und Verbesserungspotential

Obwohl die Programmierung des Dobot AI-Starters für die Linienverfolgung und Hindernisvermeidung erfolgreich war, gibt es Themen, die angegangen werden müssen um den AI-Starter zu verbessern.

Mögliche Verbesserungen

Verbesserungen bei der Linienverfolgung: In der Praxistests tritt das Problem auf, dass die Infrarotsensoren nicht immer konsistent reagieren. Selbst wenn sich das Fahrzeug vollständig auf einer weißen Fläche befindet, melden einige der Sensoren fälschlicherweise `1`, als ob sie eine schwarze Linie erkannt hätten. Dies führt zu Fehlfunktionen bei der Linienverfolgung, da der Roboter auf Basis dieser falschen Erkennung versucht zu korrigieren, obwohl keine Linie vorhanden ist. Ohne eine Möglichkeit, die Empfindlichkeit der Sensoren anzupassen, bleibt dieses Problem jedoch bestehen. (Oder man versucht die Sensoren manuel auszutauschen und befestigt neue.)

Probleme bei der Farberkennung: Auch die Farbsensoren des AI-Starters arbeiten nicht immer zuverlässig. Besonders während der Fahrt kann es vorkommen, dass Farben wie Rot oder Grün nicht korrekt erkannt werden. Dieses Problem tritt hauptsächlich auf, wenn der Roboter sich schnell bewegt, was darauf hinweist, dass die Erfassungszeit der Farbsensoren möglicherweise zu kurz ist. Eine mögliche Verbesserung wäre hier, die Sensordaten häufiger zu lesen oder die Geschwindigkeit des Roboters zu reduzieren, um eine stabilere Erkennung der Farben zu gewährleisten.

Verbesserungspotenzial bei der Hindernisvermeidung: Die Implementierung der Hindernisvermeidung funktioniert grundsätzlich, jedoch könnten die Ultraschallsensoren effizienter arbeiten. Manchmal werden Hindernisse erst spät erkannt oder kleine Objekte gar nicht. Eine mögliche Verbesserung wäre die Nutzung weiterer oder präziserer Sensoren, um eine genauere Hinderniserkennung und -vermeidung zu gewährleisten. Ein Algorithmus, der die Bewegungen bei Annäherung an Hindernisse besser optimiert, könnte ebenfalls die Reaktionszeit und Sicherheit des Roboters erhöhen.

Verbindungsprobleme mit dem XBee-Modul: Ein weiteres Problem trat bei der Verwendung des XBee-Moduls auf. Obwohl das Modul korrekt installiert und konfiguriert war, reagierte es nicht wie erwartet und konnte keine stabile Verbindung herstellen.

Trotz der erfolgreichen Implementierung bleiben diese Schwächen bestehen. Durch weitere Anpassungen an der Hardware sowie eine Optimierung der Software ließe sich der AI-Starter robuster und zuverlässiger machen.

Fazit

Im Verlauf dieses Projekts konnten wir wertvolle Erfahrungen mit der Arduino IDE und der Programmierung von Robotern sammeln. Obwohl die vorgegebenen Codes nicht wie erwartet funktionierten, entwickelten wir eigene kreative Lösungen, die gut liefen, aber noch Raum für Verbesserungen bieten, wie in den vorherigen Kapiteln beschrieben. Für alle technischen Details und die von uns entwickelten Programme verweisen wir auf das GitHub-Repository, das eine umfassende Dokumentation bereithält.

8 Checkliste

- ZeroGPT tests (Dienstag)

- Quellen und Hyperlinks etc fertig machen (Dienstag/Mittwoch)