

Aufgabe 02 - Robot Controller

Einleitung

In diesem Seminar beschäftigen wir uns mit der Programmierung von einfachen Controllern für mobile Roboter. Um das Verhalten von mobilen Robotern zu steuern, werden programmierbare Controller verwendet. Dabei spielt der Controller ein definiertes Verhalten (Programm) ab und folgt (meist) dem Sense-Plan-Act Paradigma (siehe Figure 1):

- liest Sensorwerte der Umgebung (**Sense**)
- plant den nächsten Schritt (**Plan**)
- und führt diesen aus in dem die Aktuatoren angesteuert werden (**Act**)



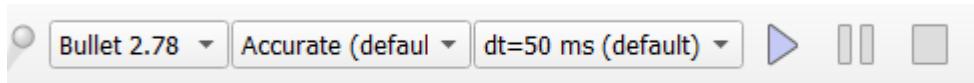
Figure 1: Sense-Plan-Act Paradigma

Es werden also die erfassten Sensorwerte verarbeitet und der nächste auszuführende Schritt geplant. Nachdem dieser Schritt geplant (und berechnet) wurde, wird eine Aktion auf den Aktoren des Roboters ausgeführt (beispielsweise die Ansteuerung von Motoren).

Die Programmierung erfolgt in der Simulationsumgebung CoppeliaSim und über dessen Python API.

Setup

1. CoppeliaSim Edu installieren: Installieren Sie die Coppelia Simulationsumgebung. Laden Sie dazu die Version 4.0 von CoppeliaSim Edu (<https://www.coppeliarobotics.com/>) herunter und installieren Sie diese lokal. **Achtung:** Bitte Version 4.0 verwenden (und nicht etwa 4.4) da sich die API in späteren Versionen geändert hat.
2. Entwicklungsumgebung einrichten: Laden Sie die Datei RobotControllerTemplate.zip aus Ilias herunter und richten Sie dieses in Ihrer Entwicklungsumgebung ein. Sie brauchen dazu Python in der Version 3.x.
3. Roboter Modelle und Simulationsszenen laden: Laden Sie die Roboter Modelle und Simulationsszenen aus Ilias herunter.
4. Simulationsszene starten: Öffnen Sie eine Simulationsszene in Coppelia und starten Sie die Simulation.



Die Simulation ist erfolgreich gestartet wenn Sie im Output eine entsprechend Ausgabe sehen.

Simulation started.

Input Lua code here, or type "help()" (use TAB for auto-completion)

5. Roboter Controller starten: Starten Sie das Programm minControllerTemplate.py. Nach der erfolgreichen Verbindung sollten Sie folgende Ausgabe erhalten: Connected to remote API server of Vrep with clientID: 0

Simulations-Szenen

In der in Ilias verfügbaren Zip-Datei sind mehrere Szenen für die Simulationsumgebung Coppelia enthalten. Diese Szenen können verwendet werden um unterschiedliche Verhalten bzw. Controller des Roboters ePuck zu testen.

Die Szene *ePuckBasicS5* (siehe Figure 2) ist eine minimalistische Szene in der ein Roboter platziert ist. Diese Szene eignet sich um sich mit der Simulationsumgebung und der Programmierung des Roboters vertraut zu machen. Eine mögliche Anwendung ist:

1. Fahren zur Wand und Stoppen (abhängig von Sensorwerten)
2. Drehung nach rechts bis nur noch der linke Sensor anschlägt
3. Geradeaus weiterfahren

Es muss keine perfekte Drehung stattfinden. Es geht darum Sensorwerte verarbeiten zu können und entsprechende Reaktionen auf diese setzen zu können. Eine Implementierung dieses Verhaltens ist als Bang-Bang und Proportional-Controller möglich.

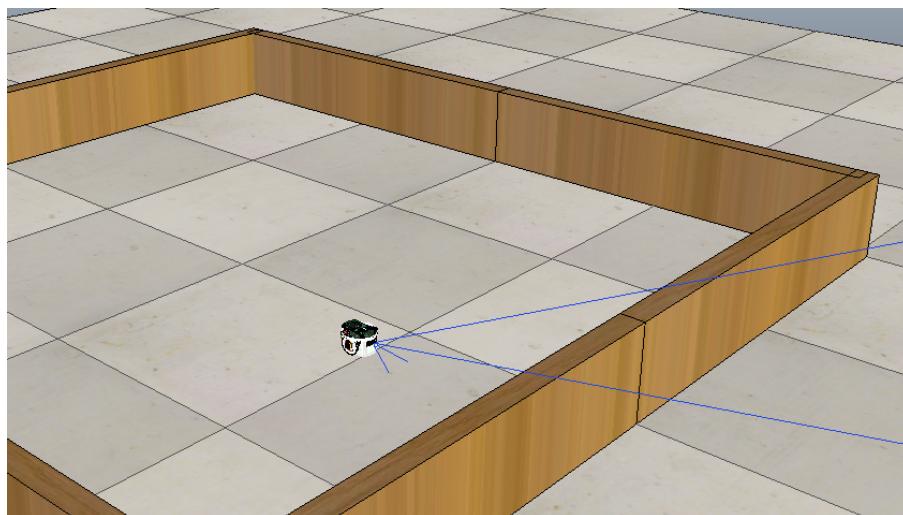


Figure 2: Basis-Szene ePuckBasicS5

Die Szene *ePuckBasicS5Braitenberg* (siehe Abbildung 3) enthält zusätzlich zum Robotermode noch einige Zylinder als Hindernisse. Diese Szene eignet sich daher sehr gut um das Verhalten eines Braitenberg-Vehikels zu testen. Dabei soll der Roboter einmal Objekten ausweichen und einmal auf diese zusteuren (Furcht- und Aggressor-Verhalten). Der Roboter darf an Objekten oder Wänden „hängen“ bleiben. Es geht darum das grundsätzliche Verhalten zu zeigen.

Achtung: Die *noDetectionDistance*-Variable muss für diese Szene angepasst werden da der Roboter anders konfigurierte Distanzsensoren verwendet.

$$noDetectionDistance = 0.5 * \text{robot.get}S()$$

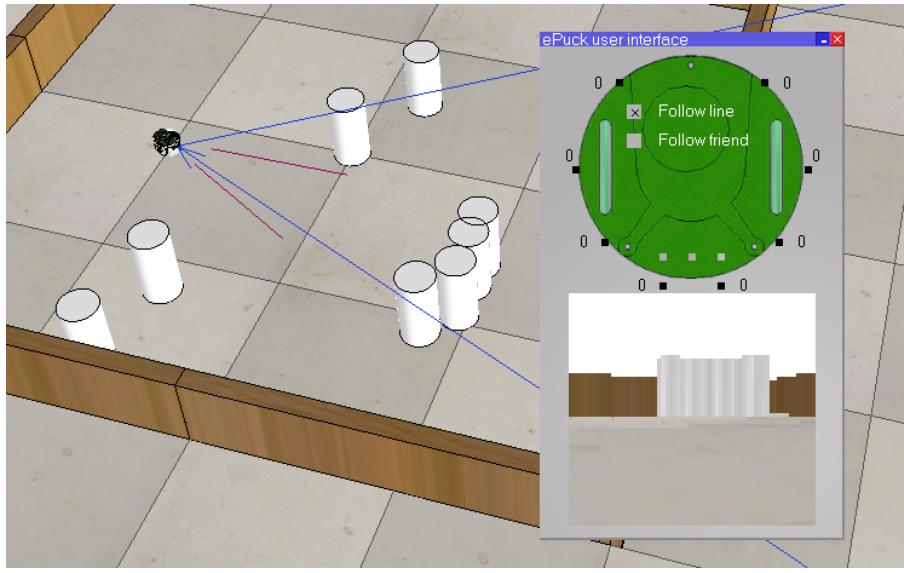


Figure 3: Szene für Braatenberg-Controller (ePuckBasicS5Braitenberg)

Die Szene *ePuckPushBox* (siehe Figure 4) enthält einen Puck der vom Roboter bewegt werden kann. Zusätzlich enthält der Roboter eine Kamera über die Bilder verarbeitet werden können. Die Bewegungen des Roboters werden in dieser Szene über einen Graphen (siehe Figure 4 rechts unten) dargestellt.

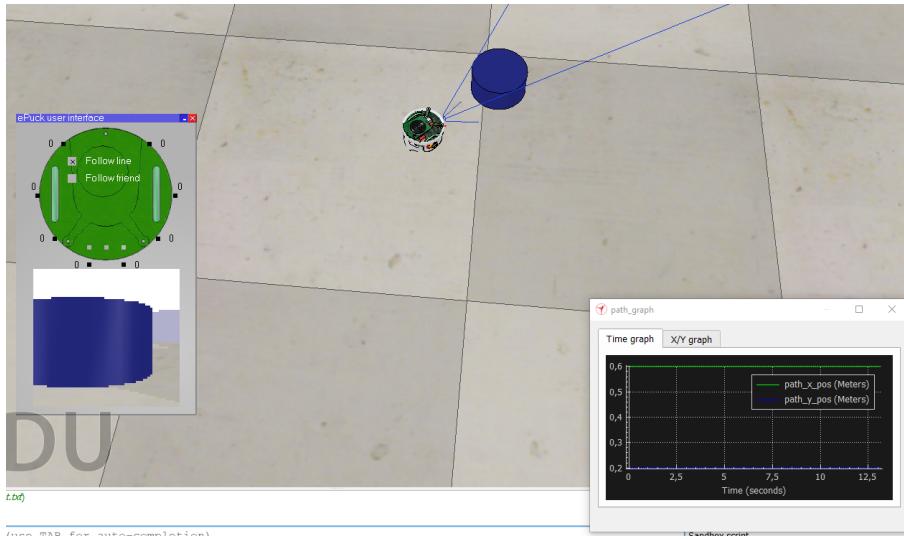


Figure 4: Szene mit bewegbarem Puck (ePuckBushBox)

Das Python-Projekt in Ilias enthält einen Controller, der zeigt wie Kamerabilder verarbeitet werden können. Die derzeitige Implementierung gibt True als Rückgabewert zurück, wenn ein Objekt detektiert wurde, ansonsten wird False zurückgegeben. Zusätzlich wird auch noch das Zentrum des erkannten Objektes bestimmt. Diese Koordinate kann als Input für einen Controller verwendet werden, um das Verhalten des Roboters anzupassen.

Der Detektionsalgorithmus ist dabei auf ein Rot-Schwarz-Bild ausgelegt. Daher muss vor der Verarbeitung des Bildes diese noch konvertiert werden. Die Konvertierung erfolgt direkt in Coppelia und betrifft die Kamereaeinstellungen des Roboters.

1. Öffnen Sie dazu die Kamereaeinstellungen



2. Fügen Sie das folgende Code-Fragment in Zeile 12 ein:

```
local trig ,packedPacket = simVision.blobDetectionOnWorkImg(
                           inData.handle , 0.300000, 0.000000, true)
if trig then
  retVal.trigger=true
end
if packedPacket then
  retVal.packedPackets[#retVal.packedPackets+1]=packedPacket
end
```

3. Wenn Sie jetzt die Szene starten, sollten Sie ein Rot-Schwarzes Bild (Schwarz ist der Puck) erhalten (siehe Figure 5).

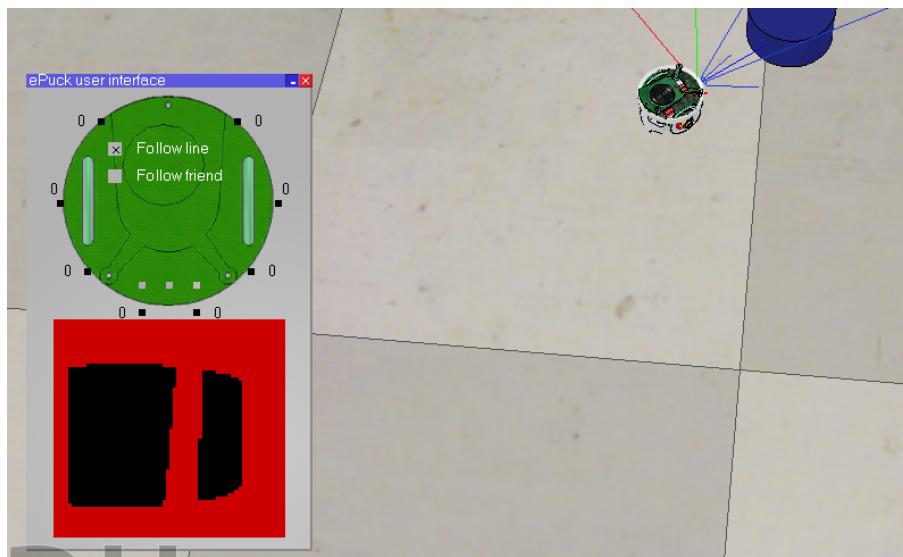


Figure 5: Vorbereitetes Kamerabild

Ein mögliches Verhalten könnte folgendes sein:

1. Puck detektieren
2. Auf Puck zusteuern
3. Puck an die Wand schieben

Subsumption Architecture

Bei einer Subsumption Architektur wird das Gesamtverhalten eines Roboters in mehrere Verhaltensteile aufgeteilt. Jedes dieser Verhalten ist dabei eigenständig und beeinflusst andere Verhalten nicht. Diese Verhalten können je nach Komplexität auch hierarchisch angeordnet sein und höherwertige Verhalten können andere Verhalten ersetzen (*subsumieren*).

Für jedes einzelne Verhalten wird zuerst geprüft, ob dieses anwendbar ist. Ist dies der Fall, wird das Verhalten ausgeführt. Im nächsten Schritt erfolgt wieder eine Überprüfung aller Verhalten und das nächste Verhalten wird ausgewählt. Sind mehrere Verhalten möglich, muss eine Reihung erfolgen. Dies kann beispielsweise durch eine Priorisierung erfolgen oder durch eine fixe Reihenfolge vorgegeben sein.

Die Szene ePuckPushMultipleBoxes (siehe Figure 6) enthält mehrere bewegbare Pucks und eignet sich für die Implementierung einer Subsumption Architecture mit mehreren Verhalten.

Das Ziel ist es Pucks an die Wand zu schieben und dann mit dem nächsten fortzufahren. Die Verhalten, welche der Roboter dabei anbieten muss, werden in einer Subsumption Architektur abgebildet. Der Roboter enthält eine Kamera über die Bilder verarbeitet werden können. Die Bewegungen des Roboters werden über einen Graphen (siehe Figure 6 rechts unten) dargestellt.

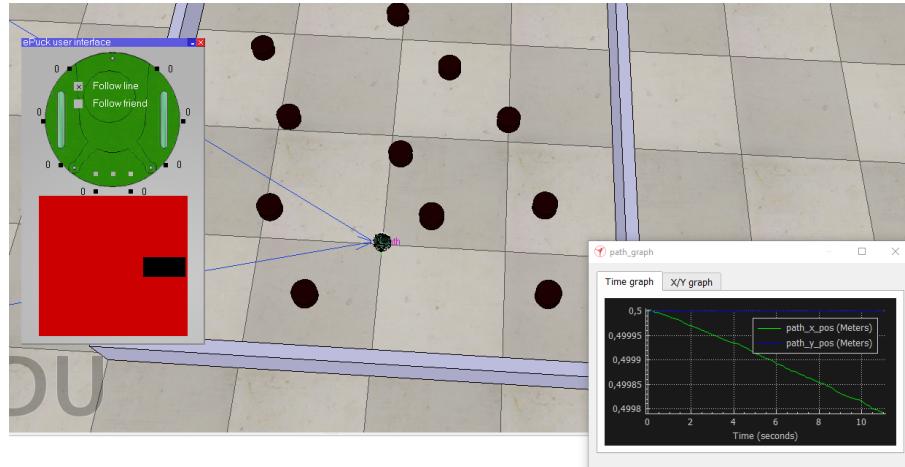


Figure 6: Szene mit mehreren bewegebaren Pucks (ePuckPushMultipleBoxes)

Ein Puck kann mittels des Kamerabilds detektiert werden. Verwenden Sie dazu die Methoden aus dem Template *RobotControllerTemplate* in Ilias. Wenn Sie keinen Puck über das Kamerabild finden, müssen Sie ein Verhalten haben, welches erlaubt einen Puck zu suchen. Hier können Sie z.B. mit einer Drehung des Roboters arbeiten. Sobald ein Puck detektiert wurde, kann darauf zugesteuert werden. Haben Sie den Puck erreicht, können Sie diesen schieben. Schieben Sie den Puck solange bis Sie eine Wand erreichen. Ob Sie eine Wand erreicht haben, können Sie mittels den *robot.getAccelerometerValues()* feststellen. Beachten Sie hierbei ob sich die Werte nur kurzzeitig oder tatsächlich für mehrere Ticks/Steps nicht ändern. Ist dies der Fall, müssen Sie sich vom Puck lösen (wegdrehen). Nachdem Sie sich vom Puck gelöst haben, setzen Sie wieder mit dem Suchen des nächsten Pucks fort.

Der Roboter zeichnet seinen gefahrenen Pfad in der Simulationsumgebung auf (siehe Figure 7).

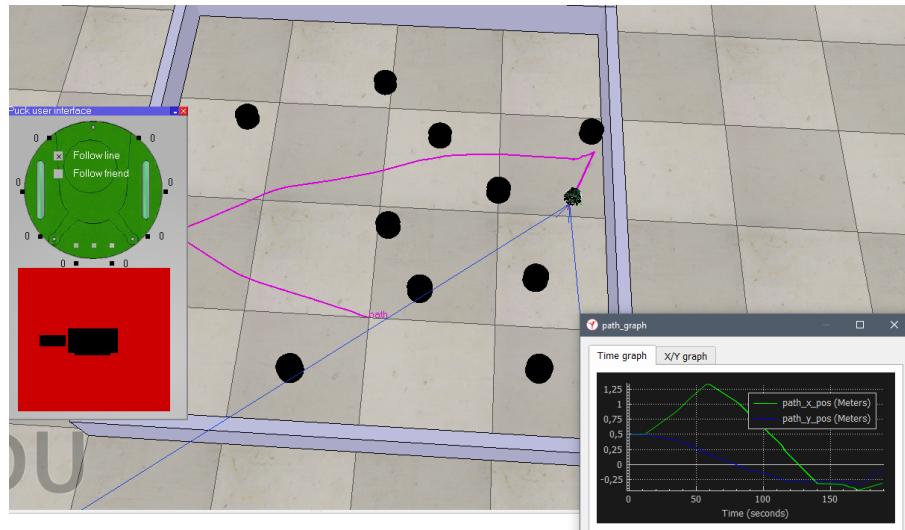


Figure 7: Pfadaufzeichnung in der Simulationsumgebung

Das Gesamtverhalten (alle Pucks müssen an eine Wand geschoben werden) ist in mehrere Verhalten aufgeteilt. Diese Verhalten sind die oben beschriebenen:

1. Puck suchen/detektieren
2. Auf Puck zusteuern
3. Puck an die Wand schieben
4. Von Puck lösen

Für jedes dieser Verhalten muss also geprüft werden, ob es im derzeitigen Zustand bzw. abhängig von den Sensorwerten angewendet werden kann. Beispielsweise kann das Verhalten „Von Puck lösen“ nur angewendet werden wenn sich die `robot.getAccelerometerValues()` für mehrere Zeitschritte nicht ändern.

Die Reihenfolge bzw. Priorisierung der Verhalten ist entscheidend. Beispielsweise wird das (Basis-)Verhalten „Puck suchen/detektieren“ in jedem Zustand möglich sein da es keine besonderen Abhängigkeiten gibt. Dieses Verhalten sollte also eine niedrigere Priorität als andere Verhalten haben. Überlegen Sie sich in welcher Reihenfolge Sie die Verhalten prüfen müssen.

Die technische Umsetzung kann auf mehrere Arten erfolgen. So kann zum Beispiel für jedes Verhalten eine Überprüfung in einer Methode und eine Ausführung in einer zweiten Methode implementiert werden. Beiden Methoden müssen dann die notwendigen Werte (Sensorwerte) übergeben werden. Im Main-Loop werden dann der Reihe nach die Prüfmethoden aufgerufen und das passende Verhalten ausgeführt. Eine alternative ist die Bündelung der notwendigen Methoden in einer Klasse. Beispielsweise könnte eine Basisklasse erstellt werden, welche die grundlegenden Methoden definiert (z.B. `isapplicable(...)` und `calculateMotorValues(...)`). Die Verhalten werden dann als abgeleitet Klassen ausgeführt und überschreiben diese Methoden entsprechend. In der Main-Loop erfolgt dann eine Überprüfung mittels `isapplicable(...)` und im Erfolgsfall wird die dazugehörige `calculateMotorValues(...)` Methode aufgerufen.

Achtung: Die Übergabeparameter der Methoden müssen in diesem Fall für alle Methoden gleich sein. Sie werden potenziell auch noch andere Methoden für eine vollständige Umsetzung brauchen.

Aufgabe

Start: 2023-04-11 Ende: 2023-05-02

Ein ePuck-Roboter soll seinen Weg durch ein einfaches Labyrinth finden. Der Roboter muss dabei zuerst den Eingang zum Labyrinth finden und sich nach erfolgreicher Durchquerung zu einem Zielobjekt hinbewegen. Implementieren Sie dieses Verhalten als Subsumption Architecture in CoppeliaSim. Die Ausarbeitung der Übung kann in 2er-Gruppen erfolgen.

Beschreibung

Ein ePuck-Roboter soll seinen Weg durch ein einfaches Labyrinth finden. Der Roboter muss dabei zuerst den Eingang zum Labyrinth finden und sich nach erfolgreicher Durchquerung zu einem Zielobjekt hinbewegen. Zu diesem Zweck muss der Roboter mehrere Verhalten unterstützen können. Beispiele für diese Verhalten sind: Lokalisieren eines Objekts, Anfahren dieses Objekts, einer Wand folgen und schließlich sich von der Wand lösen und auf ein Zielobjekt zufahren. Die Szene ist in Figure 8 dargestellt.

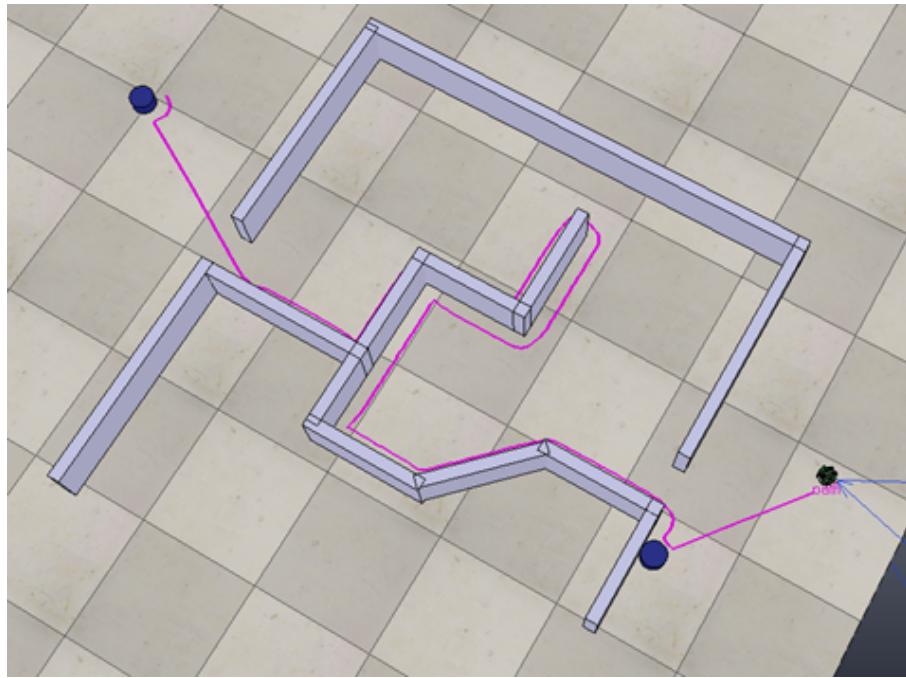


Figure 8: Labyrinth mit Zielobjekt (ePuckLabyrinth)

Der Eingang des Labyrinths kann durch ein Markerobjekt detektiert werden. Dieses Markerobjekt kann mit Hilfe der Kamera des ePuck identifiziert werden. Nach erfolgreicher Detektion muss der Roboter dieses Objekt anfahren und somit das Labyrinth betreten. Anschließend verfolgt der Roboter die Wand bis er das Zielobjekt per Kamera detektiert und sich zu diesem hinbewegen kann. Versuchen Sie die Umsetzung als Proportional Controller umzusetzen. Ein beispielhafter Pfad des Roboters ist in Figure 8 dargestellt. Nach Erreichen des Zielobjektes ist keine weitere Aktion notwendig (der Roboter darf beispielsweise das Objekt umkreisen, sich von diesem weg bewegen oder einfach stehen bleiben).

Abgabe

Die Abgabe erfolgt in Ilias bis 2023-05-02. Geben Sie den Source-Code und einen Screenshot Ihrer Implementierung ab.