Ludwig-Maximilians-Universität München
Institut für Informatik
Lehrstuhl für Mobile und Verteilte Systeme
Prof. Dr. Claudia Linnhoff-Popien

mobile and distributed systems group

Praktikum Autonome Systeme
Wintersemester 2020/21
Übungsblatt 2 – Monte Carlo Planning

Ziel der heutigen Praxisveranstaltung ist die Implementierung und Evaluation von Monte Carlo Tree Search (MCTS), womit ein Agent ein einfaches Navigationsproblem lösen soll.

Aufgabe 1: Monte Carlo Tree Search (MCTS)

Laden Sie für dieses Übungsblatt das ZIP-Archiv `autonome-systeme-uebung2.zip` runter. In diesem Archiv finden Sie die aktualisierten Quellcode-Dateien aus dem Übungsblatt 1. Zusätzlich finden Sie die Datei `multi_armed_bandits.py`, in der verschiedene Multi-armed Bandit Algorithmen implementiert sind.

Implementieren Sie einen Monte Carlo Tree Search (MCTS) Planner. Orientieren Sie sich am Gerüst der Klassen `MonteCarloTreeSearchPlanner` und `MonteCarloTreeSearchNode`.

`MonteCarloTreeSearchPlanner` ist bereits vollständig implementiert und muss zur Ausführung nur noch in der `main.py` angelegt werden.

Für die Implementierung der MCTS-Phasen, müssen Sie die Klasse `MonteCarloTreeSearchNode` folgendermaßen anpassen:

1. Implementieren Sie die Methode `select`, welche Aktionen basierend auf den aktuellen `Q_values` und `action_counts` im aktuellen Knoten auswählt. Verwenden Sie dazu die UCB1 Implementierung aus `multi_armed_bandits.py`.
2. Implementieren Sie die Methode `expand`, welche einen neuen Kindknoten erzeugt und an die Liste der aktuellen Kindknoten `self.children` anhängt.
3. Implementieren Sie die Methode `rollout`, welche `self.horizon-depth` Schritte mit einer Random-Policy simuliert, den discounted return des Rollouts berechnet und zurückgibt.

Lassen Sie den MCTS Planner über mehrere *Rooms* Episoden für verschiedene Simulationen `N` und Horizons `H` laufen. Vergleichen Sie die Performance von MCTS mit Monte Carlo Rollout Planning aus dem Übungsblatt 1.

Wie verhält sich der MCTS, wenn die UCB1 Auswahlstrategie durch Random, ϵ-Greedy (mit $\epsilon \in \{0.1, 0.25, 0.5\}$) bzw. Boltzmann Exploration (mit $\tau \in \{0.1, 1, 10\}$) ersetzt wird? Verwenden Sie dazu die Implementierungen aus `multi_armed_bandits.py`.

Zusatz:
Implementieren Sie eine "lernende" Variante von `MonteCarloTreeSearchPlanner`, die sich den ausgewählten Teilbaum merkt. Dazu müssen sie die Variable `root` als Attribut speichern.

Überschreiben Sie dazu die Methode `update` der Superklasse Agent, um das `root`-Attribut mit dem ausgewählten Teilbaum zu aktualisieren. Vergleichen Sie den "lernenden" `MonteCarloTreeSearchPlanner` mit der ursprünglichen Implementierung und mit Monte Carlo Rollout Planning.