

Versuch 8: Zustandsautomat, Liniensuchen und Hindernisse

Bearbeite alle Aufgaben schriftlich bis auf den Code. Lade den Code bis Mittwoch, den 26.06.2024, 9:00 Uhr morgens im ILIAS hoch.

Bereite dich darauf vor, den Code im Praktikum kurz vorzustellen. Benenne die Dateien in folgendem Format:

„code_08_vorname_nachname.c/h“

Der Parcours ist in mehrere Teile aufgeteilt. Im ersten Teil muss eine feste Trajektorie abgefahren werden. Daraufhin wird eine Linie verfolgt. Weiterhin gibt es eine Lücke in der Linie, die überwunden werden muss. Zusätzlich steht ein Hindernis auf der Linie, das umfahren werden muss.

1. State Machine

Der Armuro kann sich während des Rennens in verschiedenen Situationen befinden. Diese Situationen sollen als Zustände in einem Zustandsautomaten abgebildet werden. Folgende Situationen sind möglich:

- Der Armuro fährt eine feste Trajektorie ab.
(Zustand `followTrajectory` mit `task_followTrajectory()`)
- Der Armuro befindet sich auf der Linie und folgt dieser.
(Zustand `followLine` mit `task_followLine()`)
- Der Armuro hat die Linie verloren und sucht sie wieder.
(Zustand `searchLine` mit `task_searchLine()`)
- Der Armuro hat eine Lücke erkannt und sucht das gegenüberliegende Linienende.
(Zustand `overcomeGap` mit `task_overcomeGap()`)
- Der Armuro umfährt das Hindernis auf der Linie.
(Zustand `avoidObstacle` mit `task_avoidObstacle()`)

Der Zustandsautomat besteht aus Zuständen und Transitionen. Der Armuro befindet sich immer

in exakt einem Zustand. Innerhalb eines Zustands ist ein Algorithmus bzw. Regler aktiv, der den Armuro steuert. Dieser Algorithmus kann wiederum als (Sub-) Zustandsautomat mit eigenen Zuständen implementiert werden. Transitionen werden ausgelöst, wenn im aktuellen Zustand eine Übergangsbedingung erfüllt wird. Skizziere den Zustandsautomaten mit den oben genannten Zuständen sowie allen sinnvollen Transitionen, und beschrifte diese.

2. Linienverlust und Lücken erkennen

Das Abkommen von der Linie beispielsweise in einer 90° Kurve und eine Lücke sind nicht zu unterscheiden. Deshalb muss eine aktive Liniensuche eingesetzt werden, um die beiden Fälle zu unterscheiden. In dieser Aufgabe soll ein einfacher Suchalgorithmus implementiert werden, der diese Aufgabe übernimmt. Im Folgenden ist ein Verfahren beschrieben, das zur Orientierung dient. Die ersten beiden Schritte sind noch Teil von `task_followLine()`, die restlichen Schritte gehören zu `task_searchLine()`.

- Ein Linienverlust wird erkannt (abhängig vom Algorithmus zum Verfolgen der Linie).
- Beide Räder werden gestoppt.
- Das rechte Rad wird soweit gedreht bis der Armuro sich um ca. 100° gedreht hat oder die Linie wiedergefunden wird. Wurde die Linie detektiert, fährt der Armuro mit dem Linienfolgen fort (`transitionToFollowLine`).
- Wurde keine Linie detektiert, wird das rechte Rad rückwärts gedreht, bis der Armuro wieder in der Ausgangsposition steht.
- Die Schritte 3 und 4 werden gegebenenfalls mit dem linken Rad wiederholt.
- Wurde sowohl rechts als auch links keine Linie detektiert, befindet sich der Armuro wieder in der Ausgangsposition und beginnt mit dem `overcomeGap`-Task (`transitionToOvercomeGap`).

Optional: Die initiale Drehrichtung beim Liniensuchen kann auf Basis der letzten Sensorwerte ermittelt werden und muss nicht zwingend nach links sein (Heuristik).

2.1 Entwirf einen Zustandsautomaten, der alle Transitionen und Zustände des zugehörigen `searchLine`-Tasks enthält.

2.2 Implementiere den kooperativen Task `task_searchLine()` gemäß der Überlegungen in 2.1.

3. Umfahren eines Hindernisses

Implementiere den kooperativen Task `task_avoidObstacle()`, um das Hindernis auf der Linie zu umfahren.