Pure Pursuit und RRT

auf der F1/10 Plattform

J. Greten, J. Bahn, S. Fleischmann, K. Pöschel

Institute for Electrical Engineering in Medicine (UZL)

1. Juli 2020

Zusammenfassung

Verwendete Software

- ROS mit C++
- Skeleton-Code von der University of Pennsylvania für Simulation
- Visualisierung mit rviz

Die Karte

- Generierung durch Hector SLAM

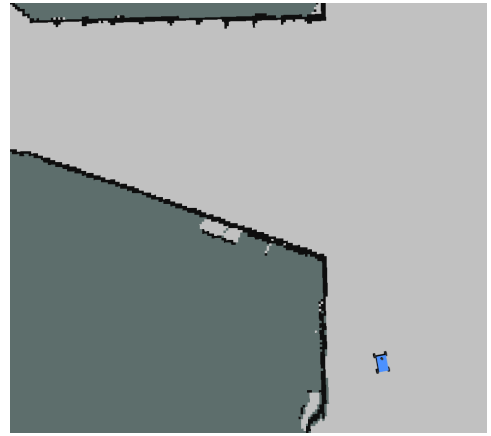

Abbildung 1: normale Karte

- Vergrößerte Hindernisse für RRT

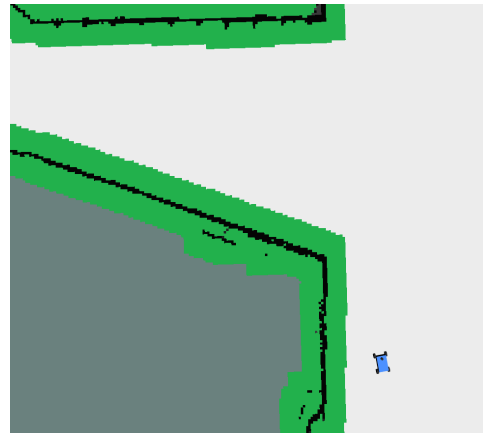

Abbildung 2: Karte mit vergrößerten Kanten

RRT

- Erzeugung eines zufälligen Punkts q_{rand}.
- Suche nach $q_{nearest}$ in bestehendem Graphen.
- Erstelle q_{new} in Richtung $q_{nearest}$ in festgelegter Schrittlänge.
- Wenn kein Hindernis zwischen $q_{nearest}$ und q_{new} liegt, füge q_{new} dem Graphen hinzu.

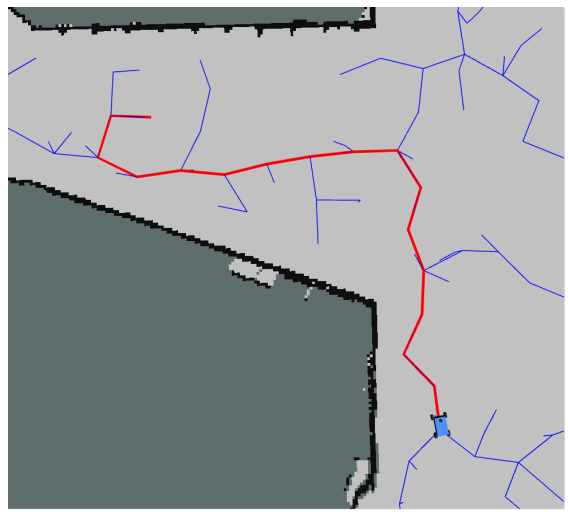

Abbildung 3: RRT

RRT

Hürden bei der Umsetzung:

- Visualisierung
- Kollisions-Überprüfung
- Menge an Parametern

RRT*

- Erweiterung von RRT.
- Nach Erstellung von q_{new} wird innerhalb einer Nachbarschaft geprüft, ob die Kosten über $q_{nearest}$ kleiner sind als die Kosten über die anderen erreichbaren Knoten und füge q_{new} mit dem Knoten mit den geringsten Kosten als Elternknoten dem Graphen hinzu.
- Überprüfe alle Knoten der Nachbarschaft, ob die Kosten mit q_{new} als Elternknoten geringer sind und aktualisiere die Verwandschaftsverhältnisse nach den optimalen Kosten.

RRT*

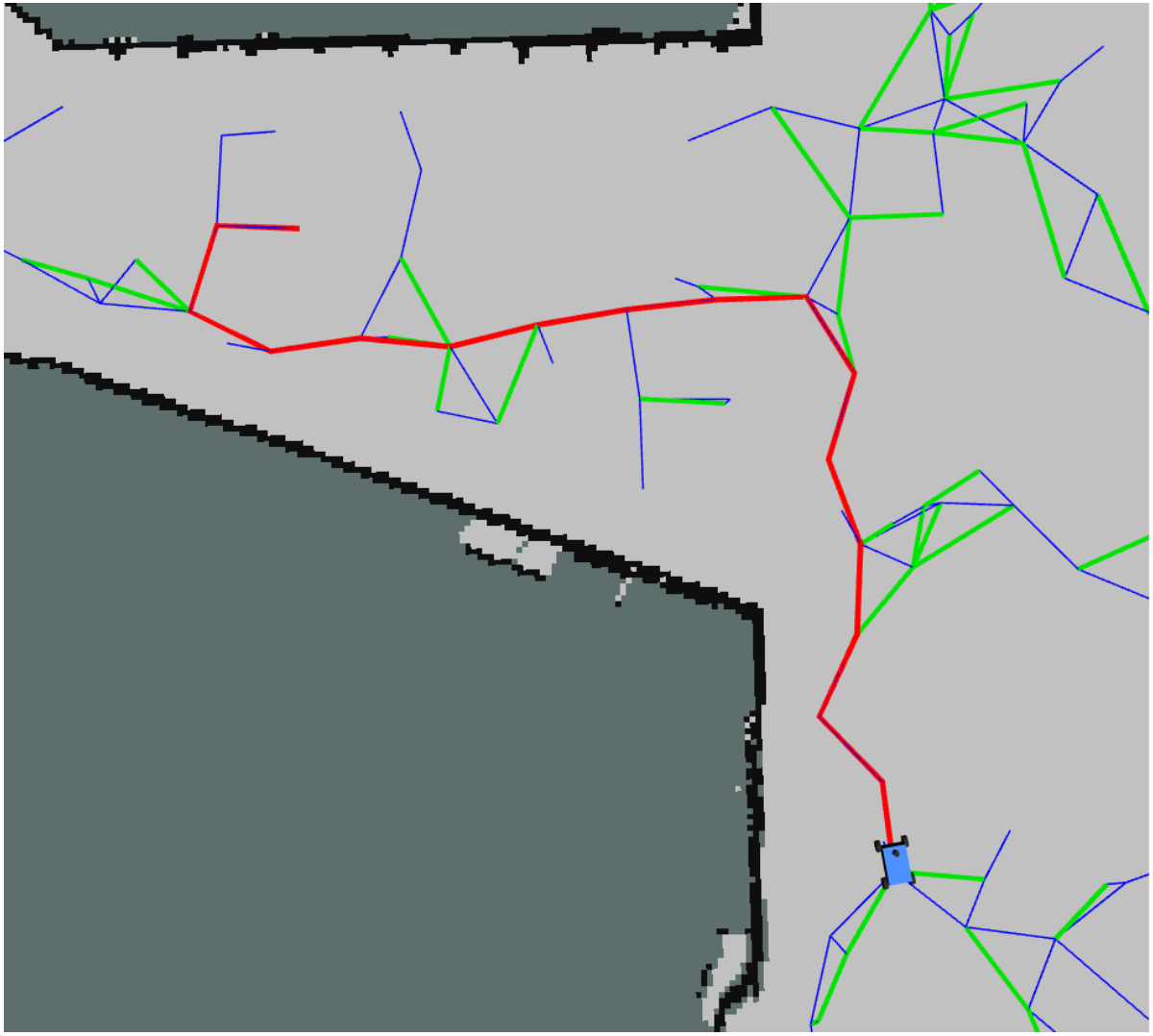

Abbildung 4: RRT*

RRT*

Hürden bei der Umsetzung:

- Definition der Nachbarschaft (Radiusberechnung) $min(\eta \cdot (log(n)/n) \cdot 1/2, step\_length)$.
- Laufzeitkomplexität

Pure Pursuit

- Controller zum Fahren des Pfades
- Suchen von einem Punkt P auf Trajektorie, der eine *lookahead distance* vom Auto entfernt ist
- Kreisbogen berechnen, der sowohl auf Punkt P und Position des Autos liegt
- Glättung der Trajektorie beim Abfahren

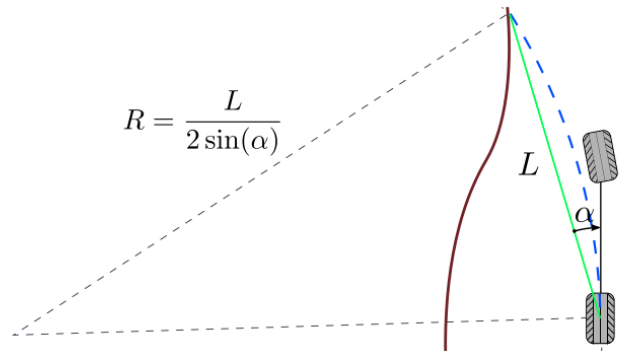

Abbildung 5: Geometrie eines Pure Pursuit Controllers

Quelle: Paden et al (2016)

Berechnung vom Look-Ahead-Punkt

- Berechnung zwischen welchen beiden RRT-Nodes der Punkt liegt
- Berechnung der möglichen Positionen von P
- Verwende die Position, die näher am Ziel ist

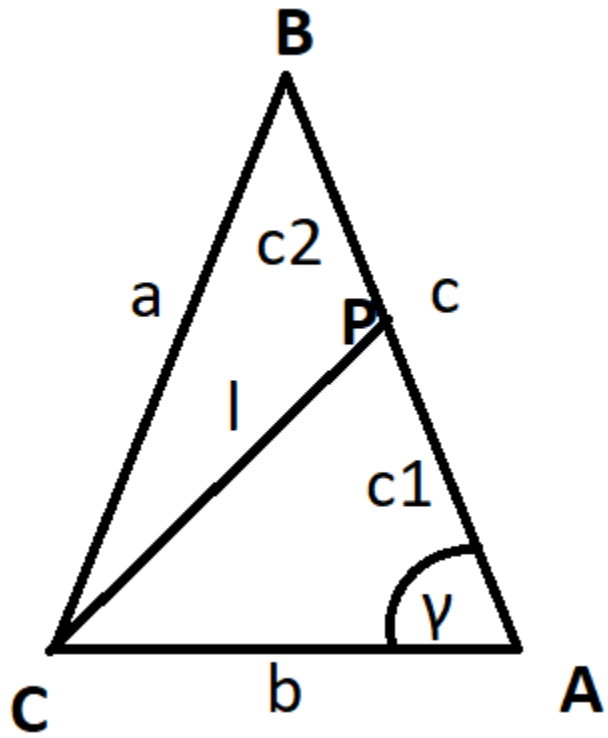

Abbildung 6: Berechnung vom Look-Ahead-Punkt

$$c_1 = b\cos(\gamma) \pm \sqrt{b^2(cos(\gamma)^2 - b^2 + l^2}$$

Umsetzung innerhalb der Simulation

Nun folgt eine kurze Präsentation.

Umsetzung außerhalb der Simulation

- Vicon Motion Capture System für Position
- Zusätzliche ros packages aus älterem Skeleton-Code

Fazit

- RRT/RRT* und Pure Pursuit arbeiten gut zusammen
- Die Laufzeitkomplexität unserer Implementierung von RRT* ist mindestens quadratisch.
- Die Struktur der Daten ist nicht optimal für die benötigte Funktionalität.
- Geschwindigkeitsberechnung in der Bahnplanung könnte zu besseren Ergebnissen führen

Quellen und Code

Code:
`https://github.com/JakobGreten/f110_pure_pursuit_rrt`

Quellen:
`https://github.com/mlab-upenn/f110-fall2019-skeletons`

`https://github.com/cyphyhouse/f1tenth_code`

`https://github.com/mlab-upenn/f110-fall2018-skeletons`

`https://github.com/mlab-upenn/f110_rrt_skeleton`

Bildquelle und Literatur:
Paden et al (2016) - A Survey of Motion Planning and Control Techniques for Self-driving Urban Vehicles

Ende

Fragen