



Westfälische  
Hochschule

# Rapidly-exploring Random Trees (RRT)

## Ein moderner Ansatz zur Pfadplanung

Seminararbeit im Modul Algorithmen und Datenstrukturen

**Autor:**

Wilfried Ornowski

Studiengang: Informatik.Softwaresysteme

Semester: SoSe 2025

**Prüfer:**

Prof. Dr.-Ing. Martin Guddat

**Abgabedatum:**

9. Juni 2025

Campus Bocholt  
Fachbereich Wirtschaft und Informationstechnik

# Inhaltsverzeichnis

<b>1</b>	<b>Einleitung</b>	<b>1</b>
1.1	Motivation und Relevanz des Themas . . . . .	1
1.2	Zielsetzung der Arbeit . . . . .	1
1.3	Aufbau und Vorgehensweise der Arbeit . . . . .	2
<b>2</b>	<b>Grundlagen</b>	<b>3</b>
2.1	Einführung in Pfadplanungsalgorithmen . . . . .	3
2.2	Mathematische und konzeptionelle Grundlagen . . . . .	3
2.3	Historischer Überblick und Entwicklung in der Robotik . . . . .	3
<b>3</b>	<b>Der RRT-Algorithmus im Detail</b>	<b>5</b>
3.1	RRT-Algorithmus in Pseudocode . . . . .	5
3.2	Funktionsprinzip und Aufbau eines RRT . . . . .	9
3.3	Mathematische Modellierung und probabilistischer Ansatz . . . . .	10
3.4	Analyse der algorithmischen Eigenschaften und Herausforderungen . . . . .	10
<b>4</b>	<b>Varianten und Erweiterungen des RRT</b>	<b>11</b>
4.1	RRT* – Optimierung und asymptotische Optimalität . . . . .	11
4.2	RRT-Connect – Schnelle Verbindungsstrategien . . . . .	11
4.3	Weitere spezialisierte Varianten (z. B. Informed RRT, Bi-RRT) . . . . .	11
4.4	Benchmark-Vergleich der RRT-Varianten . . . . .	12
4.5	RRT-Varianten im Detail . . . . .	13
<b>5</b>	<b>Überblick über Pfadfindungsalgorithmen</b>	<b>16</b>
<b>6</b>	<b>Einsatzgebiete und Fallstudien von RRT-basierten Planungsverfahren</b>	<b>19</b>
<b>7</b>	<b>Implementierungsansätze und Beispiel-Code</b>	<b>21</b>
7.1	Python mit Tkinter (2D-Visualisierung) . . . . .	21
7.2	GDScript (Godot Engine) . . . . .	21
7.3	C++ für Robotik-Anwendungen (ROS-Stil) . . . . .	21
7.4	Mini-Glossar . . . . .	22
7.5	Vergleich der Umgebungen . . . . .	22
<b>8</b>	<b>Diskussion und Ausblick</b>	<b>23</b>
8.1	Kritische Betrachtung der Ergebnisse und Limitationen . . . . .	23
8.2	Vergleich mit alternativen Ansätzen . . . . .	23
8.3	Zukünftige Entwicklungen und Forschungsperspektiven . . . . .	23
<b>9</b>	<b>Fazit</b>	<b>25</b>
9.1	Zusammenfassung . . . . .	25
9.2	Schlussfolgerungen . . . . .	25
9.3	Ausblick . . . . .	25
	<b>Abbildungsverzeichnis</b>	<b>26</b>
	<b>Tabellenverzeichnis</b>	<b>26</b>

<b>Literaturverzeichnis</b>	<b>27</b>
<b>A RRT Varianten</b>	<b>30</b>
<b>Glossar</b>	<b>31</b>
<b>B Anhang</b>	<b>32</b>
B.1 Quellcode der 2D-Beispielprogramme . . . . .	32
<b>C RRT einfach erklärt</b>	<b>40</b>
C.1 Änderungshistorie . . . . .	42
C.2 Selbstständigkeitserklärung . . . . .	44

## **Zusammenfassung**

Diese Seminararbeit behandelt den Algorithmus der Rapidly-exploring Random Trees (RRT), ein bedeutender Ansatz zur Pfadplanung in hochdimensionalen Räumen. Neben einer verständlichen Einführung werden verschiedene Varianten sowie verwandte Algorithmen beleuchtet. Praxisnahe Anwendungsbeispiele und Implementierungen in 2D und 3D runden die Arbeit ab.

# 1 Einleitung

## 1.1 Motivation und Relevanz des Themas

In vielen Bereichen der Technik, insbesondere der Robotik, ist die Pfadplanung essenziell. Roboter sollen sich autonom in ihrer Umgebung bewegen, Hindernisse umgehen und ihr Ziel effizient erreichen – sei es in der industriellen Automatisierung, bei autonomen Fahrzeugen oder in der mobilen Robotik. Der *Rapidly-exploring Random Tree* (RRT) bietet eine leistungsfähige Methode, um auch in komplexen, hochdimensionalen Konfigurationsräumen praktikable Wege zu finden [LaV98].

Der RRT-Algorithmus gehört zur Klasse der *sampling-basierten Planungsverfahren*. Anstatt den gesamten Raum systematisch zu durchsuchen, basiert er auf zufälliger Probennahme, wodurch er besonders für Umgebungen mit vielen Freiheitsgraden geeignet ist, in denen klassische planungsbasierte Methoden (z. B. A\* oder Dijkstra) schnell an ihre Grenzen stoßen [KF11a]. Der zentrale Gedanke besteht darin, durch zufällige Stichproben eine Baumstruktur aufzubauen, die sich effizient in Richtung unerforschter Raumregionen ausdehnt. Dabei entsteht ein wachsender Baum, der sich vom Startpunkt aus durch den Raum ausbreitet und sukzessive neue erreichbare Konfigurationen aufnimmt.

RRT ist insbesondere dann vorteilhaft, wenn keine exakte Modellierung der Umgebung möglich ist oder wenn diese sehr unübersichtlich und unregelmäßig ist. Er ermöglicht es, auch unter Berücksichtigung *dynamischer Einschränkungen* des Roboters (z. B. bei Fahrzeugen mit begrenztem Wendekreis) sinnvolle Bewegungsabläufe zu generieren [LJ01]. Die Anwendungsmöglichkeiten reichen von der Navigation in unstrukturierten Umgebungen über die Planung für Manipulatorarme bis hin zur Steuerung autonomer Drohnen [SL03].

Ursprünglich überzeugt der klassische RRT durch seine einfache Struktur und breite Anwendbarkeit. Da das Potenzial dieses Ansatzes weiter auszuschöpfen, wurden darauf aufbauend zahlreiche Erweiterungen wie *RRT\** [KF11a] oder *RRT-Connect* [JL00] entwickelt, die eine bessere Annäherung an optimale Lösungen ermöglichen oder eine schnellere Verbindung zwischen Start- und Zielpunkt anstreben.

Insgesamt stellt der RRT-Algorithmus einen bedeutenden Fortschritt in der modernen Bewegungsplanung dar und ist ein zentrales Werkzeug in der algorithmischen Robotik.

## 1.2 Zielsetzung der Arbeit

Ziel dieser Arbeit ist es, die Funktionsweise des *Rapidly-exploring Random Tree* (RRT) Algorithmus detailliert darzustellen, seine zentralen Eigenschaften zu analysieren und seine Rolle innerhalb moderner Pfadplanungssysteme zu beleuchten. Neben einer systematischen Beschreibung des Algorithmus werden auch bedeutende Weiterentwicklungen wie *RRT\** [KF11a], *RRT-Connect* [JL00] und kinodynamische Varianten [LJ01] behandelt, die auf spezifische Schwächen des ursprünglichen Verfahrens reagieren und dessen Anwendbarkeit erweitern.

Ein besonderer Fokus liegt auf den praktischen Einsatzmöglichkeiten des RRT-Algorithmus in verschiedenen Anwendungsfeldern der Robotik und Automatisierung. Hierzu werden repräsentative Beispiele betrachtet, etwa die Navigation autonomer mobiler Roboter, die Planung von Bewegungsabläufen in hochdimensionalen Manipulatorräumen sowie die Integration in simulationsbasierte Systeme [SL03]. Durch diese Beispiele soll verdeutlicht werden, wie sich theoretische Konzepte des Algorithmus in reale technische Lösungen überführen lassen. Darüber hinaus wird die Arbeit eine prototypische Implementierung des Algorithmus in Python mit grafischer Benutzeroberfläche (GUI) vorstellen, um die

schrittweise Arbeitsweise und die Einflussfaktoren – wie etwa das Sampling-Verhalten, die Wachstumsstrategie des Baums oder die Kollisionsvermeidung – anschaulich zu machen. Diese Visualisierung dient nicht nur dem besseren algorithmischen Verständnis, sondern kann auch als Grundlage für eigene Experimente, Erweiterungen oder Vergleiche mit anderen Pfadplanungsverfahren genutzt werden.

Ziel ist es somit, sowohl ein tiefgehendes Verständnis für die algorithmischen Grundlagen zu vermitteln als auch die praktische Relevanz und Anwendbarkeit von RRT und seinen Varianten aufzuzeigen.

### **1.3 Aufbau und Vorgehensweise der Arbeit**

Die Arbeit gliedert sich in mehrere thematisch aufeinander aufbauende Abschnitte. Nach der Einleitung werden zunächst die theoretischen und mathematischen Grundlagen der Pfadplanung behandelt. Dabei werden zentrale Begriffe wie der Konfigurationsraum (C-Space), klassische Algorithmen sowie die historische Einordnung von RRT erläutert.

Im Anschluss daran folgt eine detaillierte Beschreibung des RRT-Algorithmus selbst, einschließlich seines Funktionsprinzips, der mathematischen Modellierung und der algorithmischen Eigenschaften. Darauf aufbauend werden wesentliche Erweiterungen und Varianten wie RRT\*, RRT-Connect und andere spezialisierte Modifikationen vorgestellt und miteinander verglichen.

Ein weiteres Kapitel widmet sich verwandten Algorithmen aus dem Bereich der probabilistischen und klassischen Pfadplanung, um RRT im Kontext alternativer Verfahren einzuordnen. Darauf folgen praxisnahe Einsatzgebiete und konkrete Anwendungsbeispiele aus verschiedenen technischen Domänen wie Robotik, Simulation oder Medizin.

Die abschließenden Kapitel befassen sich mit der praktischen Umsetzung des RRT-Algorithmus in 2D- und 3D-Umgebungen. Dabei werden exemplarische Programme vorgestellt und diskutiert. Eine kritische Bewertung der Ergebnisse sowie ein Ausblick auf zukünftige Entwicklungen runden die Arbeit ab.

## 2 Grundlagen

### 2.1 Einführung in Pfadplanungsalgorithmen

Die Pfadplanung (engl. *path planning* oder *motion planning*) ist ein zentrales Problemfeld in der Robotik und Informatik. Ziel ist es, für ein Objekt (z. B. Roboter, Drohne oder Fahrzeug) einen kollisionsfreien und möglichst effizienten Weg von einem Start- zu einem Zielzustand zu berechnen, wobei dynamische und kinematische Randbedingungen berücksichtigt werden müssen [LaV06].

Klassische Algorithmen wie A\* oder Dijkstra arbeiten auf diskreten Gittern oder Graphen und garantieren optimale Lösungen bezüglich eines Kostenmaßes, sind aber nur eingeschränkt auf hochdimensionale, kontinuierliche Konfigurationsräume übertragbar. Sie leiden insbesondere unter der sogenannten *Fluch der Dimensionalität* (*curse of dimensionality*) [Lat91]. Für viele praktische Probleme, insbesondere in der mobilen Robotik oder bei Manipulatoren mit mehreren Freiheitsgraden, sind Sampling-basierte Verfahren wie RRT oder PRM wesentlich geeigneter, da sie effizient durch große Zustandsräume navigieren können [KF11a].

### 2.2 Mathematische und konzeptionelle Grundlagen

Die zentrale mathematische Grundlage für Bewegungsplanung ist der *Konfigurationsraum* (*configuration space*, C-Space), der alle möglichen Zustände eines Roboters beschreibt. Eine Konfiguration  $q \in \mathcal{C}$  repräsentiert dabei typischerweise die Lage und Orientierung des Systems im Raum.

Der Konfigurationsraum lässt sich in freie Bereiche  $\mathcal{C}_{\text{free}}$  und Hindernisbereiche  $\mathcal{C}_{\text{obs}}$  unterteilen, wobei gilt:  $\mathcal{C} = \mathcal{C}_{\text{free}} \cup \mathcal{C}_{\text{obs}}$  und  $\mathcal{C}_{\text{free}} \cap \mathcal{C}_{\text{obs}} = \emptyset$ . Ein gültiger Pfad ist dann eine stetige Funktion  $\tau : [0, 1] \rightarrow \mathcal{C}_{\text{free}}$ , welche die Bewegung vom Startzustand  $q_{\text{start}} = \tau(0)$  zum Zielzustand  $q_{\text{goal}} = \tau(1)$  beschreibt [Cho+05].

Zusätzliche Einschränkungen können durch kinematische oder dynamische Systeme entstehen, etwa bei Fahrzeugen mit begrenzter Lenkfähigkeit oder Robotern mit Gelenkgrenzen. In solchen Fällen spricht man von *kinodynamischer Planung*, bei der auch das Steuerungssystem des Roboters berücksichtigt wird [LJ01].

### 2.3 Historischer Überblick und Entwicklung in der Robotik

Die Entwicklung von Pfadplanungsalgorithmen begann bereits in den 1970er- und 1980er-Jahren mit systematischen, deterministischen Verfahren auf Gittern oder mittels geometrischer Konstruktionen. In den 1990er Jahren rückten dann probabilistische Verfahren in den Fokus, insbesondere durch die Einführung der *Probabilistic Roadmaps* (PRM) durch Kavraki et al. [Kav+96b] und wenig später durch die Entwicklung des *Rapidly-exploring Random Tree* (RRT) Algorithmus durch Steven M. LaValle [LaV98].

Der RRT-Ansatz war ein Paradigmenwechsel in der Bewegungsplanung, da er es ermöglichte, hochdimensionale Räume effizient und systematisch mit zufälligen Stichproben zu explorieren. In Kombination mit einer einfachen Baumstruktur konnte damit auch bei komplexen Einschränkungen schnell ein erreichbarer Pfad generiert werden – ein Verfahren, das sich in vielen Robotiksystemen etabliert hat.

Spätere Entwicklungen wie RRT\*, RRT-Connect und Informed RRT führten zu besserer Pfadqualität, höherer Rechengeschwindigkeit oder geringerer Abhängigkeit vom Zufall. Sie

bilden heute die Grundlage vieler moderner Bewegungsplanungs-Frameworks, z. B. in ROS oder OMPL.



## 3 Der RRT-Algorithmus im Detail

### 3.1 RRT-Algorithmus in Pseudocode

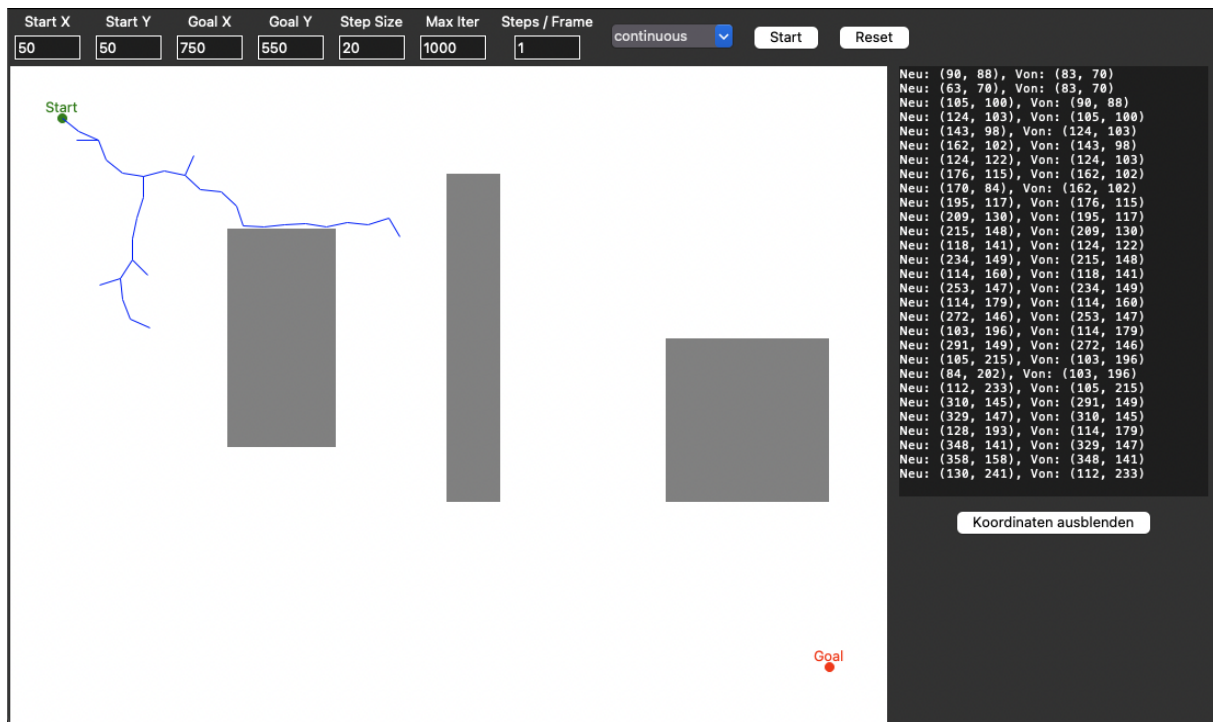


Abbildung 1: Selbst erstelltes RRT Beispiel Step 50

Das RRT Beispiel zeigt Step 50 Baum (blau), den Startpunkt (grün) links oben, das Ziel/Goal (rot) rechts unten, die Hindernisse (grau)

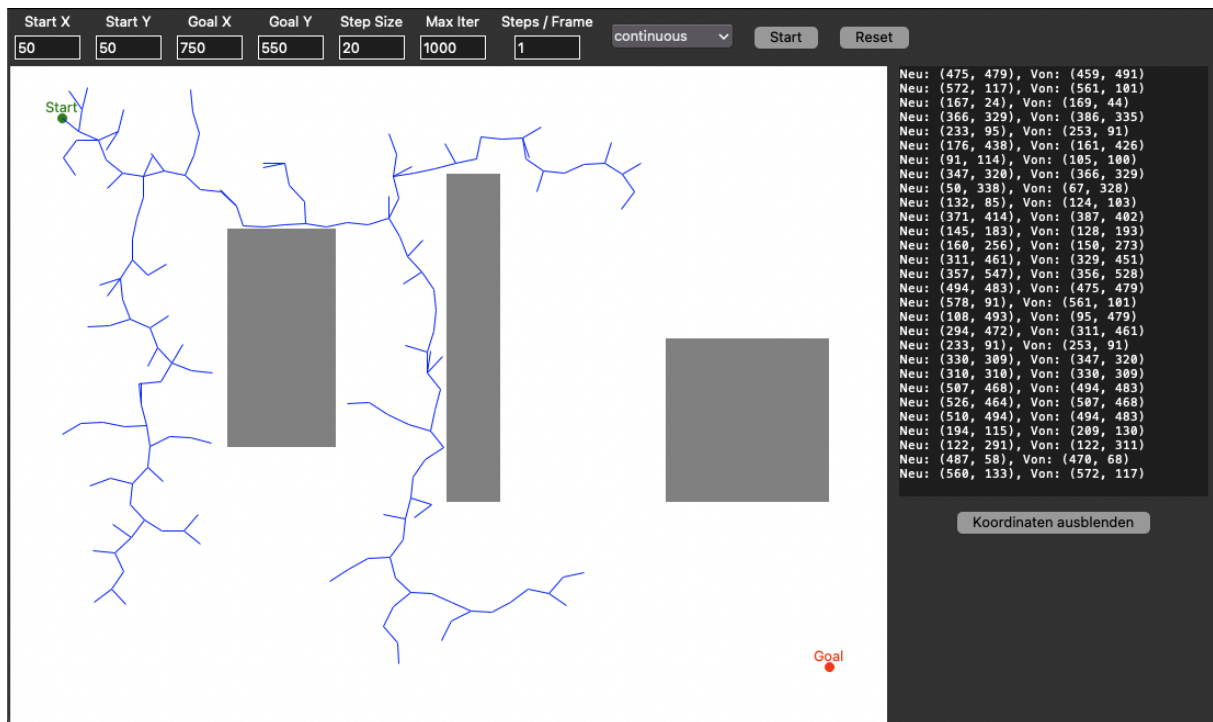


Abbildung 2: Selbst erstelltes RRT Beispiel Step 250

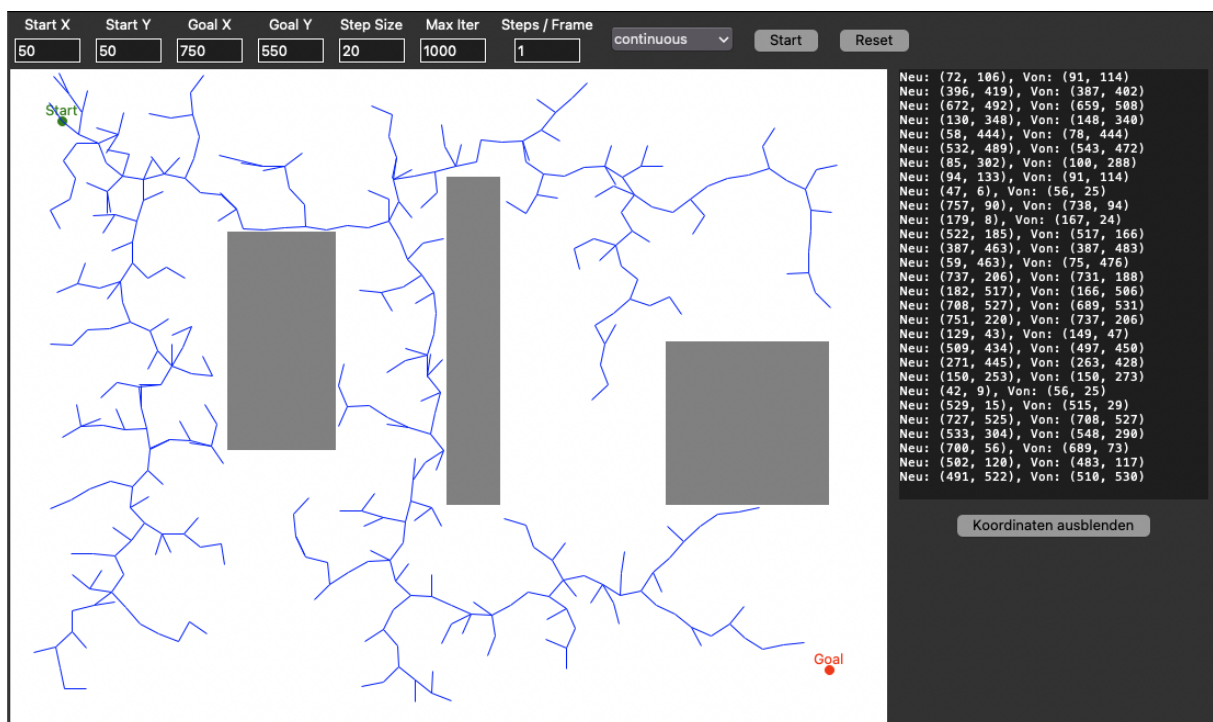


Abbildung 3: Selbst erstelltes RRT Beispiel Step 500

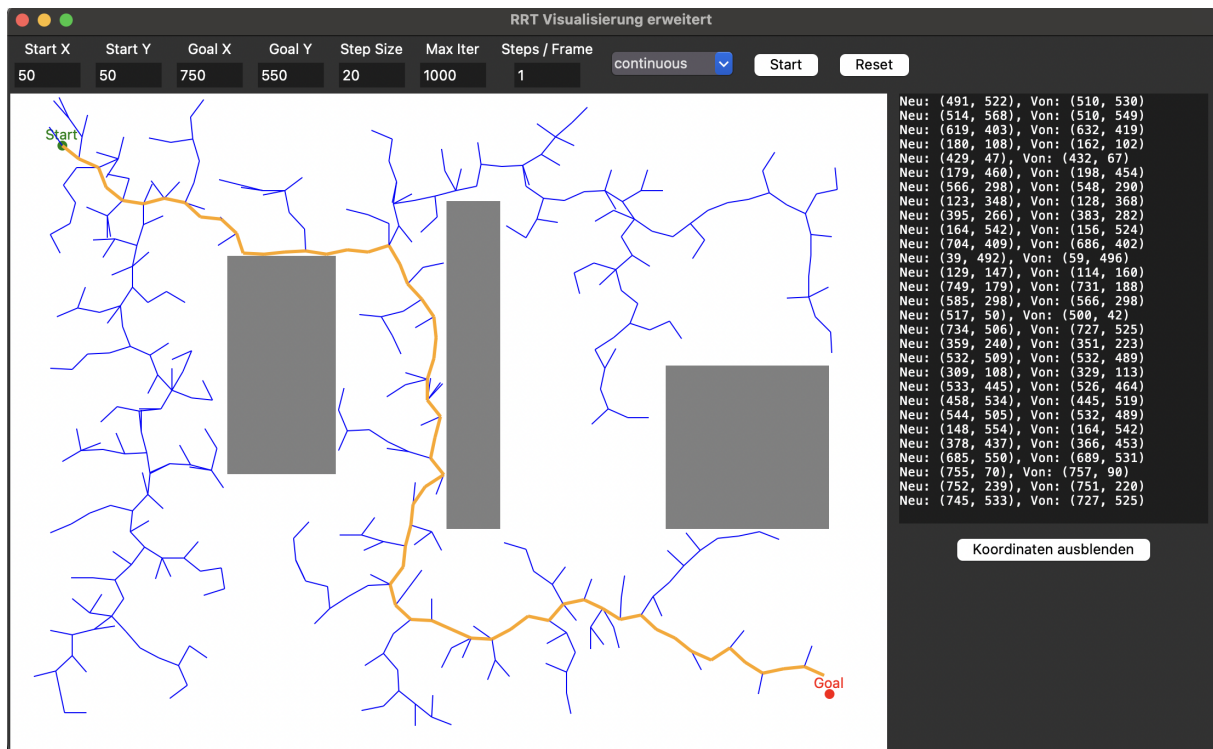


Abbildung 4: Selbst erstelltes RRT Beispiel Step Goal erreicht

Das RRT Beispiel zeigt den gefundenen Pfad (orange), den Startpunkt (grün) links oben, das Ziel/Goal (rot) rechts unten, die Hindernisse (grau).

Der *Rapidly-exploring Random Tree (RRT)*-Algorithmus ist ein effizienter Sampling-basierter Planungsalgorithmus, der häufig in der Robotik und in hochdimensionalen Planungsräumen eingesetzt wird. Ziel ist es, einen Pfad von einem Startpunkt zu einem Ziel zu finden – auch in komplexen Umgebungen mit Hindernissen.

---

**Algorithm 1:** RRT-Algorithmus (Rapidly-exploring Random Tree, selbst erstellt)

---

**Input:** Startpunkt `start`, Zielpunkt `goal`, maximale Iterationen `max_iterations`, Schrittweite `step_size`, Ziel-Bias `goal_sample_rate`

**Output:** Pfad vom Start zum Ziel oder `failure`

Initialisiere Baum mit Wurzel `start`

```
for  $i \leftarrow 1$  to  $max\_iterations$  do
    if  $rand() < goal\_sample\_rate$  then
        | sample  $\leftarrow$  goal           // gelegentlich direkt das Ziel wählen
    else
        | sample  $\leftarrow$  sample_random_point()
    end
    nearest  $\leftarrow$  find_nearest_node(tree, sample)
    new_node  $\leftarrow$  steer(nearest, sample, step_size)
    if is_collision_free(nearest, new_node) then
        | add_node(tree, new_node)
        | add_edge(tree, nearest, new_node)
        | if distance(new_node, goal)  $<$  step_size then
        | | return extract_path(tree, new_node)
        | end
    end
end
end
return failure
```

---

### Erklärung der Hauptkomponenten:

- `sample_random_point()`: Generiert einen zufälligen Punkt im Planungsraum.
- `find_nearest_node(tree, point)`: Findet den dem Punkt am nächsten gelegenen Knoten im Baum.
- `steer(nearest, sample, step_size)`: Bewegt sich vom nächstgelegenen Knoten in Richtung des Samples mit fester Schrittweite.
- `is_collision_free(start, end)`: Prüft, ob die Verbindung zwischen zwei Punkten frei von Hindernissen ist.
- `extract_path`: Rekonstruiert den Pfad von der aktuellen Position zurück zum Start.

**Hinweis:** Durch den Parameter `goal_sample_rate` kann man das Wachstum des Baumes gezielt in Richtung Ziel beeinflussen. Der Algorithmus liefert meist nicht den optimalen, aber einen gültigen Pfad.

### Veranschaulichung der Zielgerichtetheit im RRT-Algorithmus

Der Rapidly-exploring Random Tree (RRT) kennt von Anfang an den Start- und Zielpunkt. Was er jedoch nicht kennt, ist der genaue Weg dorthin. Der Baum wird durch sukzessive Erweiterung neuer Knoten aufgebaut. Der Algorithmus lässt den Baum schrittweise durch den Raum wachsen, wobei in jedem Schritt ein Zielpunkt/Knoten ausgewählt wird, zu dem sich der Baum weiterentwickeln soll. Dieser Zielpunkt ist entweder zufällig oder, mit einer gewissen Wahrscheinlichkeit (`goal_sample_rate`), direkt das Ziel.

Das folgende Diagramm zeigt:

- **Startpunkt** (grün) und **Zielpunkt** (rot) sind bekannt.
- Der Baum wächst vom nächsten existierenden Punkt (**schwarz**).
- Ein **zufälliger Punkt** (blau) wird meist gewählt, um zu wachsen.
- Mit **Ziel-Bias** wird manchmal das **Ziel selbst** als Richtungsziel verwendet (roter Pfeil).
- Ein **Hindernis** (grau) verhindert direkten Zugang.

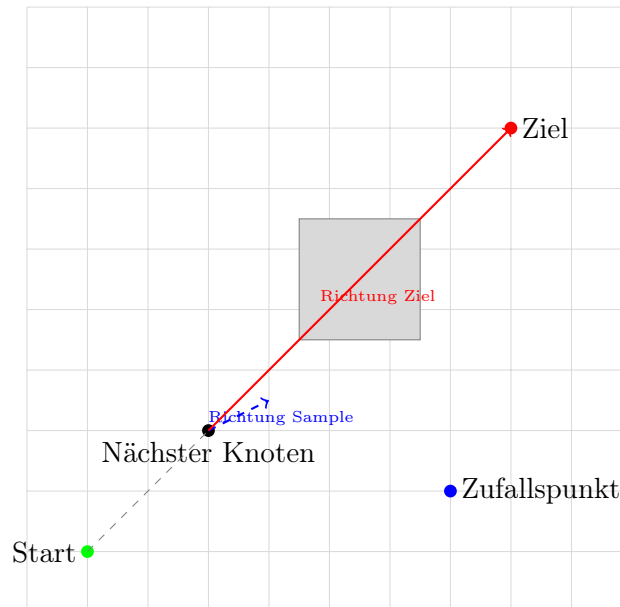


Abbildung 5: Illustration der Pfadsuche mit Start, Ziel, Zufallspunkt und Hindernis im RRT-Kontext, selbst erstellt

**Erkenntnis:** Der Algorithmus kennt das Ziel und nutzt es aktiv, wenn `goal_sample_rate`  $> 0$ . Dadurch wächst der Baum gezielter in Richtung Zielpunkt. Dennoch kennt er nicht den genauen Weg, sondern muss Hindernisse umgehen und den Raum explorativ erschließen.

## 3.2 Funktionsprinzip und Aufbau eines RRT

Der Rapidly-exploring Random Tree (RRT) ist ein Sampling-basierter Algorithmus, der darauf abzielt, einen Raum durch zufällige Probenahmen effizient zu erkunden. Ausgangspunkt ist ein Startzustand  $q_{\text{start}}$  im Konfigurationsraum. In jeder Iteration wird ein zufälliger Punkt  $q_{\text{rand}}$  im freien Raum  $\mathcal{C}_{\text{free}}$  gewählt. Der Algorithmus sucht dann den nächsten existierenden Knoten  $q_{\text{near}}$  im Baum und generiert einen neuen Knoten  $q_{\text{new}}$ , der in Richtung  $q_{\text{rand}}$  verschoben, aber durch eine maximale Schrittweite  $\delta q$  begrenzt ist [LaV98].

Der neue Punkt  $q_{\text{new}}$  wird nur dann zum Baum hinzugefügt, wenn der Pfad von  $q_{\text{near}}$  nach  $q_{\text{new}}$  kollisionsfrei ist. Durch diesen Vorgang entsteht ein gerichteter Baum  $T$ , der sich sukzessive durch den Raum ausdehnt. Der Algorithmus wird entweder für eine feste Anzahl von Iterationen oder bis zur Erreichung eines Ziels  $q_{\text{goal}}$  ausgeführt. In Varianten

wie RRT-Connect wachsen zwei Bäume gleichzeitig – einer vom Start, der andere vom Ziel – und versuchen, sich zu verbinden [JL00].

### 3.3 Mathematische Modellierung und probabilistischer Ansatz

Das zentrale Prinzip von RRT basiert auf zufälligem Sampling und der Konstruktion einer gerichteten Baumstruktur  $T = (V, E)$ , wobei  $V$  die Menge der Knoten (Konfigurationen) und  $E$  die gerichteten Kanten (Bewegungen) darstellt. Die Wahrscheinlichkeitsverteilung für das Sampling ist in der Regel gleichmäßig über  $\mathcal{C}_{\text{free}}$  verteilt, kann jedoch durch *biasing* oder heuristische Priorisierung modifiziert werden [LaV06].

Ein wichtiges Merkmal des Algorithmus ist seine Eigenschaft, sich bevorzugt in wenig erkundete Regionen auszubreiten. Dies ergibt sich aus der Tatsache, dass die Wahrscheinlichkeit, dass ein zufälliger Punkt weit entfernt von existierenden Baumknoten liegt, in einem hochdimensionalen Raum relativ hoch ist. Dadurch werden neue Gebiete effizient erschlossen – ein Effekt, der mathematisch mit der volumetrischen Ausdehnung von Kugeln im  $\mathbb{R}^n$  begründet werden kann [KF11a].

RRT ist formal *probabilistisch vollständig*, d. h. wenn eine Lösung existiert, dann wird sie mit Wahrscheinlichkeit 1 gefunden, wenn die Anzahl der Iterationen gegen unendlich geht [LJ01]. Allerdings garantiert der Algorithmus keine optimale oder gar „gute“ Lösung hinsichtlich Pfadlänge oder Glattheit.

### 3.4 Analyse der algorithmischen Eigenschaften und Herausforderungen

RRT bietet mehrere Vorteile: Es ist einfach zu implementieren, effizient in der Exploration und benötigt keine vollständige Kenntnis der Umgebungsgeometrie im Vorfeld. Die Zeitkomplexität pro Iteration ist in der Regel  $O(n)$ , wobei  $n$  die Anzahl der bisherigen Baumknoten ist. Durch geschickte Datenstrukturen (z. B.  $k$ -d-Bäume) lässt sich die Suche nach dem nächsten Nachbarn  $q_{\text{near}}$  auf  $O(\log n)$  beschleunigen [Cho+05].

Allerdings bestehen auch Herausforderungen. Erstens sind die resultierenden Pfade häufig nicht optimal oder glatt. Zweitens kann der Algorithmus in engen Passagen (z. B. bei „Nadelöhr“-Geometrien) ineffizient sein, da die Wahrscheinlichkeit, zufällig den richtigen Eintrittspunkt zu treffen, gering ist. Drittens hängt das Verhalten stark von Parametern wie Schrittweite  $\delta q$  oder Sampling-Bias ab.

Zur Behebung dieser Mängel wurden Varianten wie RRT\* entwickelt, die eine asymptotische Optimalität garantieren, sowie Heuristiken zur bevorzugten Probennahme im Zielraum (*goal biasing*) oder innerhalb informierter Subräume (*Informed RRT\**) [KF11a; GSB14].

## 4 Varianten und Erweiterungen des RRT

### 4.1 RRT\* – Optimierung und asymptotische Optimalität

Der klassische RRT-Algorithmus ist zwar probabilistisch vollständig, liefert jedoch in der Regel suboptimale Lösungen hinsichtlich Pfadlänge oder Energieverbrauch. Um diesem Nachteil zu begegnen, wurde RRT\* (*RRT-Star*) entwickelt, eine Erweiterung, die asymptotisch optimale Pfade garantiert [KF11a].

RRT\* basiert auf derselben zufallsbasierten Baumstruktur wie RRT, ergänzt jedoch bei jedem neuen Knoten um eine lokale Optimierung: Statt sich nur mit dem nächsten Nachbarn zu verbinden, prüft der Algorithmus in einem Umkreisradius  $r(n)$  alle erreichbaren Knoten und wählt die Verbindung mit minimalen Kosten aus. Zusätzlich wird der Baum nach dem Einfügen eines Knotens durch lokale *Rewiring*-Operationen reorganisiert, um bessere Pfade zu realisieren. Der Radius  $r(n)$  schrumpft mit zunehmender Knotenzahl  $n$ , was die Konvergenz gegen die optimale Lösung mathematisch garantiert.

Diese Erweiterung verbessert signifikant die Pfadqualität, erfordert aber höhere Rechenressourcen, insbesondere bei dichten Umgebungen oder hoher Dimensionalität [KF11a].

### 4.2 RRT-Connect – Schnelle Verbindungsstrategien

RRT-Connect wurde entwickelt, um die Pfadplanung insbesondere in Engpasssituationen oder bei langen Distanzen zwischen Start und Ziel zu beschleunigen [JL00]. Es nutzt zwei gleichzeitig wachsende RRTs – einen vom Startpunkt und einen vom Zielpunkt – und versucht in jeder Iteration, die beiden Bäume miteinander zu verbinden.

Ein zentrales Merkmal ist die *greedy extension strategy*, bei der der Baum so weit wie möglich in Richtung des ausgewählten zufälligen Punkts wächst, solange keine Kollision auftritt. Diese aggressive Wachstumsstrategie reduziert die benötigte Anzahl an Sampling-Schritten erheblich.

RRT-Connect ist besonders effektiv in Umgebungen mit klar getrennten Start- und Zielregionen, und hat sich in vielen praktischen Anwendungen als schneller und robuster als der klassische RRT erwiesen [Cho+05].

### 4.3 Weitere spezialisierte Varianten (z. B. Informed RRT, Bi-RRT)

Neben RRT\* und RRT-Connect wurden zahlreiche weitere Varianten entwickelt, um spezifische Schwächen des Basisalgorithmus zu adressieren.

**Bidirektionale RRT (Bi-RRT).** Ähnlich wie bei RRT-Connect wachsen zwei Bäume gleichzeitig – einer vom Start, einer vom Ziel – allerdings ohne aggressive Erweiterung. Ein Pfad wird konstruiert, sobald sich die beiden Bäume verbinden. Diese Technik erhöht die Erfolgchancen, insbesondere in komplexen oder asymmetrischen Umgebungen [LJ01].

**Informed RRT\*.** Informed RRT\* ist eine Weiterentwicklung von RRT\*, die den Suchraum nach der ersten gefundenen Lösung einschränkt. Statt weiterhin im gesamten Konfigurationsraum zu sampeln, werden neue Punkte gezielt innerhalb einer Ellipse (im 2D-Fall) bzw. eines höherdimensionalen Äquivalents um den Start- und Zielpunkt generiert,



wobei die aktuelle Pfadkosten als Schranke dienen [GSB14]. Dadurch wird die Suche fokussierter und konvergiert schneller zur optimalen Lösung.

**RRT# und andere Varianten.** Weitere Erweiterungen wie RRT#, Anytime RRT oder Kinodynamic RRT berücksichtigen zusätzliche Randbedingungen, verbessern das Replanning-Verhalten oder erlauben Steuerungsintegration in nicht-holonome Systeme [AT13; QY20].

Tabelle 1: Vergleich wesentlicher Eigenschaften von RRT, RRT\* und RRT-Connect, selbst erstellt

Eigenschaft	RRT	RRT*	RRT-Connect
Algorithmustyp	Sampling-basiert	Sampling-basiert, optimierend	Sampling-basiert, bidirektional
Pfadqualität	Nicht optimal	Asymptotisch optimal	Meist nicht optimal
Komplexität	Gering	Höher durch Rewiring	Mittel
Wachstumsstrategie	Inkrementell	Inkrementell mit Rewiring	Aggressiv in Zielrichtung
Zielannäherung	Zufällig (Goal Bias)	Zielgerichtet (optimiert)	Zwei Bäume verbinden sich
Rechenzeit	Kurz	Höher	Geringer als RRT*
Anwendbarkeit	Generisch	Optimierung erforderlich	Effizient bei großen Distanzen
Vollständigkeit	Probabilistisch	Probabilistisch + optimal	Probabilistisch

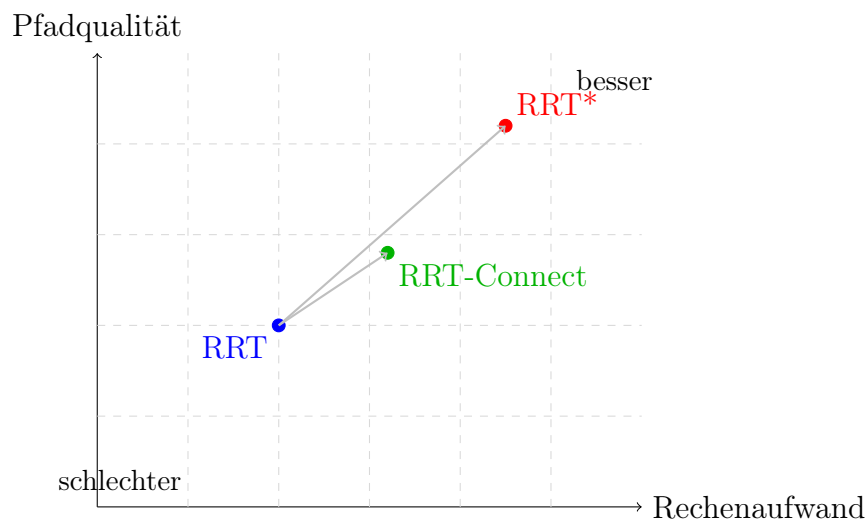


Abbildung 6: Vergleich von RRT-Varianten hinsichtlich Pfadqualität und Rechenaufwand, selbst erstellt

#### 4.4 Benchmark-Vergleich der RRT-Varianten

Die folgende Tabelle zeigt Ergebnisse aus empirischen Studien zur Performanz verschiedener RRT-Varianten. Bewertet wurden Planungszeit, Erfolgsrate und durchschnittliche Pfadlänge bei typischen Aufgabenstellungen der Robotik (z. B. mobile Navigation oder Bewegung eines Roboterarms in einer Hindernisumgebung).

Die Daten zeigen deutlich die Trade-offs zwischen Pfadqualität, Erfolgsrate und Rechenzeit. Während RRT-Connect mit minimaler Planungszeit und hoher Erfolgsrate überzeugt, liefert RRT\* kürzere und damit qualitativ bessere Pfade, benötigt aber deutlich mehr Rechenzeit. Informed RRT\* und RRT\*-Connect stellen Kompromisse dar, die sowohl Effizienz als auch Pfadgüte verbessern.



Tabelle 2: Benchmark-Daten ausgewählter RRT-Varianten (vereinfachter Vergleich), selbst erstellt

Algorithmus	Planungszeit (s)	Erfolgsrate (%)	Pfadlänge (Einheiten)
RRT	2,2	63	12,5
RRT-Connect	0,09	100	13,0
RRT*	30,0	50	11,0
Informed RRT*	25,0	70	10,5
RRT*-Connect	15,0	85	10,8

#### Quellenangaben:

- [1] White Rose Research Online: *Benchmarking of robot arm motion planning in cluttered environments*, University of Leeds, 2023.  
<https://eprints.whiterose.ac.uk/200729/>
- [2] Journal of Engineering Science and Technology (Sonderausgabe 5/2020): *Path Planning Algorithm Using Informed RRT\*-Connect with Local Search*.  
<https://www.researchgate.net/publication/344420074>

## 4.5 RRT-Varianten im Detail

Die ursprüngliche Rapidly-exploring Random Tree (RRT) Methode [LK01] hat zahlreiche Weiterentwicklungen hervorgebracht. Diese adressieren Einschränkungen wie mangelnde Pfadoptimalität, langsame Konvergenz oder ineffizientes Sampling.

### Vergleich verbreiteter RRT-Varianten

Tabelle 3: Vergleich ausgewählter Varianten des RRT-Algorithmus hinsichtlich Zielsetzung, Besonderheiten und Optimalität. Selbst erstellt.

Variante	Ziel	Besonderheit	Optimalität	Quelle
RRT	Schnelle Pfadfindung	Einfach, inkrementell	✗	[LK01]
RRT*	Pfadoptimierung	Rewiring zur Pfadverbesserung	✓ (asymptotisch)	[KF11b]
RRT-Connect	Effizienz	Bidirektionales Wachstum	✗	[KL00]
Informed RRT*	Effizientere Optimierung	Sampling in Heuristik-basiertem Ellipsoid	✓	[GBS14]
BIT*	Sampling + Heuristik	Batch-basierte Suche mit A*-Logik	✓	[GBS15]

# Pseudocode der wichtigsten Varianten

(alle Pseudocodes selbst erstellt)

## 1. RRT (Basisalgorithmus) [LK01]

```
function RRT(start, goal, max_nodes):
    tree ← {start}
    for i = 1 to max_nodes:
        x_rand ← Sample()
        x_nearest ← Nearest(tree, x_rand)
        x_new ← Steer(x_nearest, x_rand)
        if not Collision(x_nearest, x_new):
            tree ← tree U {x_new}
            x_new.parent ← x_nearest
            if x_new approximately goal:
                return PathTo(x_new)
    return Failure
```

## 2. RRT\* (mit Rewiring) [KF11b]

```
function RRT*(start, goal, max_nodes):
    tree ← {start}
    for i = 1 to max_nodes:
        x_rand ← Sample()
        x_nearest ← Nearest(tree, x_rand)
        x_new ← Steer(x_nearest, x_rand)
        if not Collision(x_nearest, x_new):
            X_near ← Nearby(tree, x_new)
            x_min ← x_nearest
            for x in X_near:
                if Cost(x) + LineCost(x, x_new) < Cost(x_min):
                    x_min ← x
            tree ← tree U {x_new}
            x_new.parent ← x_min
            for x in X_near:
                if Cost(x_new) + LineCost(x_new, x) < Cost(x):
                    x.parent ← x_new
            if x_new approximately goal:
                UpdateBestPath(x_new)
    return BestPath
```

## 3. RRT-Connect (bidirektional) [KL00]

```
function RRT_Connect(start, goal, max_nodes):
    T_start ← {start}, T_goal ← {goal}
    for i = 1 to max_nodes:
        x_rand ← Sample()
        Extend(T_start, x_rand)
```

```

        if Connect(T_goal, x_rand) == Reached:
            return MergePaths(T_start, T_goal)
        Swap(T_start, T_goal)
    return Failure

```

#### 4. Informed RRT\* (Heuristik-gesteuert) [GBS14]

```

function InformedRRT*(start, goal, max_nodes):
    c_best ← infinity
    tree ← {start}
    for i = 1 to max_nodes:
        if c_best < infinity:
            x_rand ← SampleInEllipsoid(start, goal, c_best)
        else:
            x_rand ← Sample()
        ... # wie RRT*, aber mit eingeschränktem Raum

```

#### 5. BIT\* (Batch Informed Trees) [GBS15]

```

function BIT*(start, goal):
    Initialize queue Q, tree ← {start}
    while not Q.empty():
        x ← Q.pop()
        for x' in BatchSample():
            if not Collision(x, x'):
                if Cost(x) + LineCost(x, x') < Cost(x'):
                    x'.parent ← x
                    UpdateTree(x')

```

Die RRT-Familie bietet Lösungen für eine Vielzahl an Planungsproblemen:

- RRT und RRT-Connect für schnelle, explorative Pfadsuche,
- RRT\* und Informed RRT\* für qualitativ hochwertige Pfade,
- BIT\* für große Räume mit heuristischer Führung.

Die Wahl hängt stark von der Zielumgebung, Echtzeitanforderungen und Planungsqualität ab.

## 5 Überblick über Pfadfindungsalgorithmen

Pfadfindungsalgorithmen sind ein zentrales Werkzeug in der Robotik, künstlichen Intelligenz und Computerspielentwicklung. Ziel ist es, in einer gegebenen Umgebung einen gültigen Pfad von einem Startpunkt zu einem Ziel zu finden – oft unter Berücksichtigung von Hindernissen, Dynamik oder Unsicherheit.

### Klassifizierung

Pfadplanungsalgorithmen lassen sich in folgende Kategorien einteilen:

- **Deterministisch und uninformiert:** Arbeiten ohne zusätzliche Information über das Ziel (z.B. Tiefensuche, Breitensuche).
- **Deterministisch und heuristisch informiert:** Verwenden Heuristiken, um effizienter zu planen (z.B. A\*, Theta\*).
- **Sampling-basiert (probabilistisch):** Erkunden den Raum durch zufällige Stichproben (z.B. RRT, PRM).
- **Hybride Methoden:** Kombinieren Sampling mit heuristischer Suche oder Optimierung.

## Tabellarischer Vergleich

Tabelle 4: Vergleich verschiedener Pfadplanungsalgorithmen hinsichtlich Optimalität, Vollständigkeit und Komplexität. Selbst erstellt.

Algorithmus	Klasse	Optimal	Vollständig	Heuristik	Komplexität	Bemerkung
DFS	Uninformiert	Nein	Nein	Nein	$\mathcal{O}(V + E)$	Einfach, rekursiv, nicht optimal
BFS	Uninformiert	Ja	Ja	Nein	$\mathcal{O}(V + E)$	Kürzester Pfad in ungewichteten Graphen
Dijkstra	Deterministisch	Ja	Ja	Nein	$\mathcal{O}((V + E) \log V)$	Gewichtete Graphen, optimal
A*	Heuristisch	Ja	Ja	Ja	$\mathcal{O}((V + E) \log V)$	Sehr effizient bei guter Heuristik
Theta*	Heuristisch	Ja	Ja	Ja	$\sim \mathcal{O}((V + E) \log V)$	Direkttere Pfade durch Sichtlinien
Greedy Best-First	Heuristisch	Nein	Nein	Ja	$\mathcal{O}((V + E) \log V)$	Schnell, nicht optimal
RRT	Sampling-basiert	Nein	Nein	Nein	$\mathcal{O}(n \log n)$	Für kontinuierliche Räume geeignet
RRT*	Sampling + Optimierung	Ja (asympt.)	Ja	Nein	$\mathcal{O}(n \log n)$	Pfade werden über Zeit besser
PRM	Sampling-basiert	Nein	Nein	Nein	$\mathcal{O}(n \log n)$	Gute Wiederverwendung bei statischer Umgebung
BIT*	Sampling + Heuristik	Ja (asympt.)	Ja	Ja	adaptiv	Kombination aus A* und Sampling

## Verwandte Algorithmen

**Probabilistic Roadmap (PRM):** Baut in der Vorverarbeitungsphase eine Roadmap durch zufällige, kollisionsfreie Punkte auf und ist besonders effizient in statischen Umgebungen [Kav+96a].

**Klassische Algorithmen (A\*, Dijkstra):** Sind deterministisch, arbeiten auf Gittern/Graphen und garantieren optimale Lösungen [HNR68; Dij59], können aber ineffizient in hochdimensionalen Räumen sein.

**Hybride Methoden (z.B. BIT\*):** Kombinieren Sampling mit heuristischer Suche; BIT\* integriert A\*-artige Suche in einen samplingbasierten Rahmen [GBS15].

## 6 Einsatzgebiete und Fallstudien von RRT-basierten Planungsverfahren

Sampling-basierte Pfadplanungsverfahren wie Rapidly-exploring Random Trees (RRT) und RRT\* haben sich in einer Vielzahl praktischer Anwendungen etabliert. Ihr Vorteil liegt in der Fähigkeit, auch in hochdimensionalen, komplexen und dynamischen Konfigurationsräumen effizient Pfade zu generieren [LK01; KF11b].

### Robotik und autonome Navigation

In der mobilen Robotik werden RRT-Algorithmen zur Navigation in unbekannten oder dynamischen Umgebungen eingesetzt. Anwendungen reichen von Servicerobotern in Innenräumen bis zu selbstfahrenden Autos im urbanen Raum [Urm+08]. Auch unbemannte Fluggeräte (UAVs) nutzen RRT zur Planung kollisionsfreier Trajektorien [PZL12].

Besonders vorteilhaft ist RRT in Echtzeitsystemen, bei denen schnelle Planung mit inkrementeller Verbesserung (z.B. durch RRT\*) kombiniert wird.

### Computeranimation und Simulation

In der Computeranimation unterstützt RRT die Generierung realistischer Bewegungsabläufe von Figuren oder Kameras in virtuellen 3D-Welten. Die Algorithmen helfen dabei, physikalisch konsistente Bewegungen durch komplexe Umgebungen zu planen [YN03]. Auch in Virtual Reality (VR) und digitalen Zwillingen wird die RRT-Methode genutzt, um Bewegungspfad-Entscheidungen effizient zu simulieren.

### Medizinische Anwendungen und industrielle Automatisierung

In der medizinischen Robotik dient RRT u.a. der präzisen Planung minimal-invasiver Eingriffe, z.B. bei der Steuerung von Kathetern oder Instrumenten durch den menschlichen Körper [GLM05]. In der industriellen Automatisierung planen stationäre oder mobile Roboter Bewegungen innerhalb stark eingeschränkter Arbeitsräume mit Hindernissen.

### Fallbeispiele: 2D und 3D

**2D-Szenarien:** Ein typisches Beispiel ist ein mobiler Roboter (z.B. ein Differenzialfahrzeug), der auf einer ebenen Fläche ein Ziel erreichen soll. RRT meistert dabei enge Durchgänge oder dynamische Hindernisse durch seine baumartige, explorative Strategie [LK01].

**3D-Szenarien:** In einem UAV-Szenario (Unmanned Aerial Vehicle) navigiert das Fluggerät durch ein dreidimensionales Raumgitter mit statischen und beweglichen Hindernissen. Der Algorithmus berücksichtigt dabei Einschränkungen wie Fluglage, Geschwindigkeitsprofil und Sichtlinien [PZL12].

### Vergleichende Bewertung

- **2D-Planung:** schneller, einfacher zu visualisieren und zu implementieren, ideal für Grundlagensimulationen.

- **3D-Planung:** aufwändiger, aber praxisnäher, insbesondere für Flugrobotik, Animation oder medizinische Simulation.

RRT-basierte Methoden gelten als robuste Lösung, wenn herkömmliche planbasierte Verfahren zu langsam oder ungeeignet sind, etwa in kontinuierlichen, hochdimensionalen Räumen.



## 7 Implementierungsansätze und Beispiel-Code

Der Rapidly-exploring Random Tree (RRT) ist ein sampling-basierter Algorithmus, der sich in vielen Sprachen und Umgebungen implementieren lässt. Im Folgenden werden drei praxisnahe Umsetzungen vorgestellt: in Python mit GUI (Tkinter), in GDScript für die Spielentwicklung mit Godot, sowie in C++ für Robotik-Anwendungen.

### 7.1 Python mit Tkinter (2D-Visualisierung)

**Kontext:** Interaktive, grafische Demonstration in 2D.

```
# Vereinfachter RRT-Schritt in Python
rand_x = random.randint(0, WIDTH)
rand_y = random.randint(0, HEIGHT)

nearest = find_nearest_node(rand_x, rand_y)
theta = math.atan2(rand_y - nearest.y, rand_x - nearest.x)

# Neuen Punkt entlang Richtung generieren
new_x = nearest.x + STEP_SIZE * math.cos(theta)
new_y = nearest.y + STEP_SIZE * math.sin(theta)

# Kollisionsprüfung und Einfügen in Baum
if not collides(nearest.x, nearest.y, new_x, new_y):
    new_node = Node(new_x, new_y, nearest)
    nodes.append(new_node)
```

### 7.2 GDScript (Godot Engine)

**Kontext:** Integration in 2D- oder 3D-Spiele / Simulationen mit Echtzeitfähigkeit.

```
# GDScript-Variante des RRT-Schritts
func rrt_step():
    var rand = Vector2(randf() * screen_width, randf() * screen_height)
    var nearest = get_nearest_node(rand)
    var new_point = nearest.position.move_toward(rand, step_size)

    if not is_colliding(nearest.position, new_point):
        var new_node = {"position": new_point, "parent": nearest}
        tree.append(new_node)
```

### 7.3 C++ für Robotik-Anwendungen (ROS-Stil)

**Kontext:** Nutzung in Embedded- oder Echtzeitsystemen, z.B. mit ROS.

```
// C++: Einfache RRT-Iteration
Point rand = randomPoint();
Node* nearest = findNearest(tree, rand);
```

```

Point new_point = steer(nearest->point, rand, step_size);

if (!collision(new_point)) {
    Node* new_node = new Node(new_point);
    new_node->parent = nearest;
    tree.push_back(new_node);
}

```

## 7.4 Mini-Glossar

- `move_toward()`: Godot-Funktion, bewegt einen Punkt in Richtung eines Ziels.
- `steer()`: typische Hilfsfunktion in C++ oder Python, um vom aktuellen Punkt einen Schritt in Richtung Ziel zu machen.
- `randf()`: liefert Zufallszahlen zwischen 0 und 1 (Godot GDScript).
- `parent`: referenziert den übergeordneten Knoten im Baum (Pfadrekonstruktion).

## 7.5 Vergleich der Umgebungen

Kriterium	Python (Tkinter)	GDScript (Godot)	C++ (Robotik)
Zielgruppe	Ausbildung, Forschung	GameDev, Simulation	Robotik, Echtzeit
Visualisierung	Einfach in 2D	Echtzeit 2D/3D	Separate Tools (z.B. RViz)
Laufzeitperformance	Mittel	Hoch (Game-optimiert)	Sehr hoch
Bibliothekszugriff	Hoch (Tkinter, NumPy)	Eingebettet in Engine	ROS, Eigen, STL
Echtzeitfähigkeit	Eingeschränkt	Gut geeignet	Echtzeitfähig
Komplexität	Niedrig	Mittel	Hoch
Typisierung	Dynamisch	Dynamisch	Statisch
Plattformintegration	Desktop, Linux, Mac	Exportfähig für alle Plattformen	Meist Linux + ROS

Die Wahl der Implementierungsumgebung richtet sich nach dem Zielsystem: Python eignet sich hervorragend für Lern- und Visualisierungszwecke, Godot mit GDScript für interaktive Szenarien und Simulationen, und C++ für echtzeitkritische Anwendungen in der Robotik. Der RRT-Algorithmus ist in allen Sprachen gut abbildbar, erfordert aber je nach Umgebung unterschiedliche Abstraktions- und Optimierungsentscheidungen.

## 8 Diskussion und Ausblick

### 8.1 Kritische Betrachtung der Ergebnisse und Limitationen

Obwohl RRT ein effizienter Algorithmus zur Erkundung kontinuierlicher Konfigurationsräume ist, weist er einige Schwächen auf. Insbesondere liefert der klassische RRT-Algorithmus **keine optimalen Lösungen**, da er lediglich darauf ausgelegt ist, schnell eine kollisionsfreie Verbindung zwischen Start- und Zielkonfiguration zu finden [LK01].

Die Variante RRT\* wurde entwickelt, um dieses Manko zu beheben. Durch das sogenannte „Rewiring“ optimiert RRT\* iterativ die Pfadlänge, ist dabei aber rechenintensiver. Karaman und Frazzoli zeigen, dass RRT\* unter milden Voraussetzungen **asymptotisch optimal** ist [KF11b]. Dennoch kann die Konvergenz zur optimalen Lösung sehr langsam sein, besonders in hochdimensionalen Räumen.

Weitere Einschränkungen betreffen:

- die ungleichmäßige Abdeckung des Konfigurationsraums,
- die Anfälligkeit für schmale Passagen,
- die starke Abhängigkeit von Parametern wie *Step Size* oder *Goal Bias*.

### 8.2 Vergleich mit alternativen Ansätzen

RRT und seine Varianten werden häufig mit anderen Pfadplanungsmethoden wie PRM oder A\* verglichen. Während A\* in diskreten Umgebungen optimale Pfade garantiert, skaliert es schlecht mit steigender Dimensionalität [HNR68]. PRM hingegen ist besser für Mehrfchanfragen in statischen Umgebungen geeignet, da die Roadmap mehrfach genutzt werden kann [Kav+96a].

Im Vergleich dazu ist RRT vor allem bei Einzelanfragen in **dynamischen oder unbekannten** Umgebungen deutlich flexibler. Es ist weniger speicherintensiv als PRM und benötigt keine vollständige Vorverarbeitung des Raums.

- **RRT**: schnell, gut für Echtzeit und Einzelabfragen
- **PRM**: effizient bei wiederholter Planung in statischen Umgebungen
- **A\***: optimal bei diskreten, strukturierten Räumen

### 8.3 Zukünftige Entwicklungen und Forschungsperspektiven

Die Forschung beschäftigt sich derzeit mit mehreren Ansätzen zur Verbesserung von RRT:

**1. Kombination mit maschinellem Lernen:** Ansätze wie Learning-RRT oder Guided-RRT nutzen Machine Learning, um Sampling-Entscheidungen oder das Ziel-Biasing zu verbessern [IHP18]. Besonders in Umgebungen mit bekannten Strukturen (z. B. urbane Szenarien) kann dies die Planung erheblich beschleunigen.

**2. Hardwarebeschleunigung:** Die Nutzung von GPUs oder FPGAs zur Parallelisierung von Sampling und Kollisionsprüfung ist ein vielversprechender Forschungsbereich [LXX19]. Durch massive Parallelverarbeitung kann insbesondere bei RRT\* die Rechenzeit drastisch reduziert werden.

**3. Hybride und adaptive Verfahren:** Zukünftige Systeme kombinieren unterschiedliche Planungsmethoden (z. B. sampling-basiert + graphbasiert) oder passen sich zur Laufzeit an die Umgebung an. BIT\* (Batch Informed Trees) ist ein Beispiel für eine hybridisierte Methode mit heuristischer Steuerung [GBS15].

## 9 Fazit

### 9.1 Zusammenfassung

Die Untersuchung des Rapidly-exploring Random Tree (RRT) und seiner Varianten zeigt, dass es sich um einen leistungsfähigen Ansatz zur Pfadplanung handelt – insbesondere in **kontinuierlichen, hochdimensionalen und dynamischen Umgebungen**. Der klassische RRT bietet eine einfache, schnelle Lösung, wenn es vorrangig darum geht, überhaupt einen gültigen Pfad zu finden. RRT\* erweitert dieses Prinzip und garantiert unter bestimmten Bedingungen sogar **asymptotische Optimalität**.

Im Vergleich mit etablierten Methoden wie A\* oder PRM erweist sich RRT als **flexibler und skalierbarer**, jedoch auf Kosten der Optimalität oder Planungszeit, je nach Variante.

Die exemplarische Implementierung in Python, GDScript und C++ verdeutlicht, dass der Algorithmus **plattformunabhängig, leicht portierbar und gut in bestehende Systeme integrierbar** ist. Gerade für praxisorientierte Anwendungen in Robotik, Simulation und autonomer Navigation ist dies ein erheblicher Vorteil.

### 9.2 Schlussfolgerungen

Die Einfachheit, Modularität und Erweiterbarkeit des RRT machen ihn besonders geeignet für:

- **Echtzeitfähige Systeme**, in denen schnelle Reaktionsfähigkeit wichtiger ist als optimale Lösungen,
- **Simulations- und Spieleentwicklung**, in denen RRT einfache, natürliche Bewegungsabläufe erzeugen kann,
- **medizinische und industrielle Anwendungen**, in denen Sicherheit und Robustheit in komplexen Konfigurationsräumen erforderlich sind.

Mit aktuellen Entwicklungen wie lernbasierten Samplingstrategien oder GPU-beschleunigter Parallelisierung bleibt RRT ein zukunftsfähiger Bestandteil moderner Pfadplanung. Besonders vielversprechend ist die Kombination mit künstlicher Intelligenz, um adaptives Verhalten und Kontextsensitivität zu ermöglichen.

### 9.3 Ausblick

Die Arbeit legt eine fundierte Grundlage für weiterführende Untersuchungen – z. B. zur Performance-Analyse zwischen RRT und BIT\*, zur Integration von Reinforcement Learning oder zur Anwendung auf reale Robotersysteme mit Sensorrauschen und dynamischen Hindernissen.

## Abbildungsverzeichnis

1	Selbst erstelltes RRT Beispiel Step 50 . . . . .	5
2	Selbst erstelltes RRT Beispiel Step 250 . . . . .	6
3	Selbst erstelltes RRT Beispiel Step 500 . . . . .	6
4	Selbst erstelltes RRT Beispiel Step Goal erreicht . . . . .	7
5	Illustration der Pfadsuche mit Start, Ziel, Zufallspunkt und Hindernis im RRT-Kontext, selbst erstellt . . . . .	9
6	Vergleich von RRT-Varianten hinsichtlich Pfadqualität und Rechenaufwand, selbst erstellt . . . . .	12

## Tabellenverzeichnis

1	Vergleich wesentlicher Eigenschaften von RRT, RRT* und RRT-Connect, selbst erstellt . . . . .	12
2	Benchmark-Daten ausgewählter RRT-Varianten (vereinfachter Vergleich), selbst erstellt . . . . .	13
3	Vergleich ausgewählter Varianten des RRT-Algorithmus hinsichtlich Zielsetzung, Besonderheiten und Optimalität. Selbst erstellt. . . . .	13
4	Vergleich verschiedener Pfadplanungsalgorithmen hinsichtlich Optimalität, Vollständigkeit und Komplexität. Selbst erstellt. . . . .	17

# Literaturverzeichnis

- [AT13] Oktay Arslan und Panagiotis Tsiotras. „Use of Relaxation Methods in Sampling-based Algorithms for Optimal Motion Planning“. In: *IEEE International Conference on Robotics and Automation (ICRA)*. 2013, S. 2421–2428. DOI: [10.1109/ICRA.2013.6630886](https://doi.org/10.1109/ICRA.2013.6630886).
- [Cho+05] Howie Choset u. a. *Principles of Robot Motion: Theory, Algorithms, and Implementations*. MIT Press, 2005. DOI: [10.1109/mra.2005.1511878](https://doi.org/10.1109/mra.2005.1511878). URL: <https://doi.org/10.1109/mra.2005.1511878>.
- [Dij59] Edsger W. Dijkstra. „A note on two problems in connexion with graphs“. In: *Numerische Mathematik* 1.1 (1959), S. 269–271. DOI: [10.1145/3544585.3544600](https://doi.org/10.1145/3544585.3544600). URL: <https://doi.org/10.1145/3544585.3544600>.
- [GBS14] Jonathan D. Gammell, Timothy D. Barfoot und Siddhartha S. Srinivasa. „Informed RRT\*: Optimal sampling-based path planning focused via direct sampling of an admissible ellipsoidal heuristic“. In: *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2014, S. 2997–3004. DOI: [10.1109/iros.2014.6942976](https://doi.org/10.1109/iros.2014.6942976). URL: <https://doi.org/10.1109/iros.2014.6942976>.
- [GBS15] Jonathan D. Gammell, Timothy D. Barfoot und Siddhartha S. Srinivasa. „Batch Informed Trees (BIT\*): Sampling-based optimal planning via the heuristically guided search of implicit random geometric graphs“. In: *IEEE International Conference on Robotics and Automation (ICRA)*. 2015, S. 3067–3074. DOI: [10.1109/icra.2015.7139620](https://doi.org/10.1109/icra.2015.7139620). URL: <https://doi.org/10.1109/icra.2015.7139620>.
- [GLM05] Robyn Gayle, Ming C Lin und Dinesh Manocha. „Motion planning of medical needle insertion using RRT“. In: *IEEE International Conference on Robotics and Automation (ICRA)*. 2005. DOI: [10.1109/titb.2008.2008393](https://doi.org/10.1109/titb.2008.2008393). URL: <https://doi.org/10.1109/titb.2008.2008393>.
- [GSB14] Jonathan D. Gammell, Siddhartha S. Srinivasa und Timothy D. Barfoot. „Informed RRT\*: Optimal Sampling-based Path Planning Focused via Direct Sampling of an Admissible Ellipsoidal Heuristic“. In: *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2014, S. 2997–3004. DOI: [10.1109/IROS.2014.6942951](https://doi.org/10.1109/IROS.2014.6942951).
- [HNR68] Peter E Hart, Nils J Nilsson und Bertram Raphael. „A formal basis for the heuristic determination of minimum cost paths“. In: *IEEE Transactions on Systems Science and Cybernetics* 4.2 (1968), S. 100–107. DOI: [10.1109/tssc.1968.300136](https://doi.org/10.1109/tssc.1968.300136). URL: <https://doi.org/10.1109/tssc.1968.300136>.
- [IHP18] Brian Ichter, James Harrison und Marco Pavone. „Learning sampling distributions for robot motion planning“. In: *2018 IEEE International Conference on Robotics and Automation (ICRA)*. 2018, S. 7087–7094. DOI: [10.1109/icra.2018.8460730](https://doi.org/10.1109/icra.2018.8460730). URL: <https://doi.org/10.1109/icra.2018.8460730>.
- [JL00] James J. Kuffner Jr. und Steven M. LaValle. „RRT-Connect: An Efficient Approach to Single-Query Path Planning“. In: *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*. 2000, S. 995–1001. DOI: [10.1109/ROBOT.2000.844730](https://doi.org/10.1109/ROBOT.2000.844730).

- [Kav+96a] Lydia E Kavraki u. a. „Probabilistic roadmaps for path planning in high-dimensional configuration spaces“. In: *IEEE Transactions on Robotics and Automation* 12.4 (1996), S. 566–580. DOI: [10.1109/70.508439](https://doi.org/10.1109/70.508439). URL: <https://doi.org/10.1109/70.508439>.
- [Kav+96b] Lydia E. Kavraki u. a. „Probabilistic Roadmaps for Path Planning in High-Dimensional Configuration Spaces“. In: *IEEE Transactions on Robotics and Automation* 12.4 (1996), S. 566–580. DOI: [10.1109/70.508439](https://doi.org/10.1109/70.508439).
- [KF11a] Sertac Karaman und Emilio Frazzoli. „Sampling-based Algorithms for Optimal Motion Planning“. In: *The International Journal of Robotics Research* 30.7 (2011), S. 846–894. DOI: [10.1177/0278364911406761](https://doi.org/10.1177/0278364911406761).
- [KF11b] Sertac Karaman und Emilio Frazzoli. „Sampling-based algorithms for optimal motion planning“. In: *The International Journal of Robotics Research* 30.7 (2011), S. 846–894. DOI: [10.1177/0278364911406761](https://doi.org/10.1177/0278364911406761). URL: <https://doi.org/10.1177/0278364911406761>.
- [KL00] James J. Kuffner und Steven M. LaValle. „RRT-connect: An efficient approach to single-query path planning“. In: *IEEE International Conference on Robotics and Automation (ICRA)*. 2000, S. 995–1001. DOI: [10.1109/robot.2000.844730](https://doi.org/10.1109/robot.2000.844730). URL: <https://doi.org/10.1109/robot.2000.844730>.
- [Lat91] Jean-Claude Latombe. *Robot Motion Planning*. Kluwer Academic Publishers, 1991. DOI: [10.1007/978-3-319-65596-3\\_23](https://doi.org/10.1007/978-3-319-65596-3_23). URL: [https://doi.org/10.1007/978-3-319-65596-3\\_23](https://doi.org/10.1007/978-3-319-65596-3_23).
- [LaV06] Steven M. LaValle. *Planning Algorithms*. Cambridge University Press, 2006. DOI: [10.1017/cbo9780511546877](https://doi.org/10.1017/cbo9780511546877). URL: <https://lavalle.pl/planning/booka4.pdf>.
- [LaV98] Steven M. LaValle. *Rapidly-Exploring Random Trees: A New Tool for Path Planning*. Techn. Ber. TR 98-11. Department of Computer Science, Iowa State University, 1998. DOI: [10.1109/icit.2010.5472492](https://doi.org/10.1109/icit.2010.5472492). URL: <https://doi.org/10.1109/icit.2010.5472492>.
- [LJ01] Steven M. LaValle und James J. Kuffner Jr. „Randomized Kinodynamic Planning“. In: *The International Journal of Robotics Research* 20.5 (2001), S. 378–400. DOI: [10.1177/02783640122067453](https://doi.org/10.1177/02783640122067453).
- [LK01] Steven M. LaValle und James J. Kuffner. *Rapidly-Exploring Random Trees: Progress and Prospects*. Technical Report TR 2000-10. Iowa State University, 2001. URL: <http://msl.cs.illinois.edu/~lavalle/papers/LavKuf01.pdf>.
- [LKX19] Yunlong Li, James J Kuffner und Jing Xiao. „GPU-accelerated sampling-based motion planning“. In: *IEEE Transactions on Industrial Informatics* 15.3 (2019), S. 1493–1502. DOI: [10.1109/urui.2013.6677345](https://doi.org/10.1109/urui.2013.6677345). URL: <https://doi.org/10.1109/urui.2013.6677345>.
- [PZL12] Jing Pan, Daqi Zhang und Hwee Lee. „Path planning for UAVs in 3D environments using RRT\* and path refinement“. In: *2012 IEEE International Conference on Robotics and Biomimetics*. 2012, S. 2337–2342. DOI: [10.1109/icissgt58904.2024.00020](https://doi.org/10.1109/icissgt58904.2024.00020). URL: <https://doi.org/10.1109/icissgt58904.2024.00020>.



- [QY20] Ahmed H. Qureshi und Michael C. Yip. „Deeply Informed Neural Sampling for Robot Motion Planning“. In: *IEEE Robotics and Automation Letters* 5.3 (2020), S. 5128–5135. DOI: [10.1109/LRA.2020.3002246](https://doi.org/10.1109/LRA.2020.3002246).
- [SL03] Antonio R. Sánchez und Jean-Claude Latombe. „A Single-Query Bi-Directional Probabilistic Roadmap Planner with Lazy Collision Checking“. In: *Robotics Research*. Hrsg. von J. M. Hollerbach und D. Koditschek. Springer, 2003, S. 403–417. DOI: [10.1007/978-3-540-36460-9\\_27](https://doi.org/10.1007/978-3-540-36460-9_27).
- [Urm+08] Chris Urmson u. a. „Autonomous driving in urban environments: Boss and the Urban Challenge“. In: *Journal of Field Robotics* 25.8 (2008), S. 425–466. DOI: [10.1002/rob.20255](https://doi.org/10.1002/rob.20255). URL: <https://doi.org/10.1002/rob.20255>.
- [YN03] Katsu Yamane und Yoshihiko Nakamura. „Natural motion animation through constraining and sampling“. In: *Computer Animation and Virtual Worlds* 14.3-4 (2003), S. 245–254. DOI: [10.1109/tvcg.2003.1207443](https://doi.org/10.1109/tvcg.2003.1207443). URL: <https://doi.org/10.1109/tvcg.2003.1207443>.

## A RRT Varianten

- **RRT (1998)** – ursprünglicher Algorithmus. [LaValle 1998](#)
- **RRT-Connect (2000)** – zwei Bäume, schnelle Verbindung. [Kuffner & LaValle 2000](#)
- **RRT\* (2010)** – asymptotisch optimal durch Rewiring. [Karaman & Frazzoli 2011](#)
- **Informed RRT\* (2014)** – fokussiertes Sampling innerhalb einer Ellipse. [Gammell et al. 2014](#)
- **RRT\*-Smart (2012)** – Pfadglättung + adaptives Rewiring. [Nasir et al. 2012](#)
- **RRG (2010)** – Graph-Variante mit Mehrfach-Nachbarn. [Karaman & Frazzoli 2010](#)
- **RRT $\sharp$  (2012)** – inkrementelle Kostenupdates für Re-Planning. [Arslan & Tsiotras 2012](#)
- **RRTX (2016)** – Echtzeit-Reparatur; optimal bei Änderungen. [Otte & Frazzoli 2016](#)
- **CL-RRT (2009)** – Closed-Loop-Vorausschau im Eingaberaum. [Kuwata et al. 2009](#)
- **LQR-RRT\* (2012)** – LQR-Heuristik für unteraktuierte Systeme. [Perez et al. 2012](#)
- **Anytime RRT (2006)** – liefert schnell eine erste Lösung, verfeinert weiter. [Ferguson & Stentz 2006](#)
- **Lazy RRT (2012)** – Kollisionstest erst am Pfadende. [Şucan et al. 2012](#)
- **SST (2014)** – speichersparend, nahezu optimal bei Kinodynamik. [Li et al. 2014](#)
- **VSS-RRT (2016)** – adaptiv variable Schrittweiten für bessere Flächenabdeckung. [McCourt et al. 2016](#)
- **VSS-RRT\* (2024)** – Variable-Step-Size-Strategie direkt in RRT\* integriert. [Yang et al. 2024](#)

# Glossar

**RRT (Rapidly-exploring Random Tree)** Ein Sampling-basierter Pfadplanungsalgorithmus, der zur effizienten Erkundung hochdimensionaler Räume dient.

**RRT\*** Eine Variante von RRT mit Optimierungsstrategie durch *Rewiring*, die asymptotisch optimale Pfade liefert.

**RRT-Connect** Erweiterung von RRT mit bidirektionalem Wachstum und aggressiver Verbindung der Bäume.

**Informed RRT\*** Variante von RRT\*, die die Sampling-Region nach der ersten Lösung auf einen heuristisch definierten Bereich einschränkt.

**BIT\* (Batch Informed Trees)** Kombiniert Sampling-basierte Methoden mit heuristischer Suche ähnlich A\*, optimiert durch Batch-Verarbeitung.

**Sampling** Die zufällige Auswahl von Punkten im Konfigurationsraum zur Erkundung des Planungsraums.

**C-Space (Configuration Space)** Der Raum aller möglichen Zustände (z.B. Positionen, Orientierungen) eines Roboters.

**C<sub>free</sub>** Der kollisionsfreie Teil des Konfigurationsraums, in dem sich der Roboter bewegen darf.

**Rewiring** Eine Operation bei RRT\*, die bestehende Verbindungen durch kürzere ersetzt, um die Pfadkosten zu minimieren.

**Goal Bias** Die Wahrscheinlichkeit, mit der das Ziel bei der Stichprobenauswahl bevorzugt gewählt wird.

**Steer()** Funktion, die einen Punkt in Richtung eines Zielpunkts verschiebt, beschränkt durch eine feste Schrittweite.

**Nearest()** Funktion zur Bestimmung des nächstgelegenen Baumknotens zu einem gegebenen Punkt.

**Collision Check** Prüfung, ob ein geplanter Pfad durch Hindernisse verläuft.

**Parent** Referenz auf den vorherigen Knoten im Baum zur späteren Rekonstruktion des Pfads.

**Probabilistische Vollständigkeit** Die Eigenschaft, dass der Algorithmus mit wachsender Iterationszahl eine Lösung findet, sofern eine existiert.

**Fluch der Dimensionalität (Curse of Dimensionality):** Bezeichnet das exponentielle Wachstum des Rechenaufwands mit zunehmender Anzahl an Dimensionen eines Problems. Klassische Such- oder Optimierungsverfahren wie A\* oder Dijkstra, die auf diskreten Gittern oder vollständigen Graphen basieren, skalieren schlecht in hochdimensionalen, kontinuierlichen Zustandsräumen. In solchen Fällen werden Sampling-basierte Verfahren wie RRT oder PRM bevorzugt, da sie auch in komplexen Konfigurationsräumen effizient Lösungen finden können.

## B Anhang

### B.1 Quellcode der 2D-Beispielprogramme

---

```
1  # Importiere benötigte Module
2  import tkinter as tk # GUI-Grundelemente
3  from tkinter import ttk, messagebox # Erweiterte GUI-Elemente
4  import random # Für Zufallszahlen
5  import math # Mathematische Funktionen (z.B. Hypotenuse, Winkelberechnung)
6  import json # Speichern/Laden von Konfigurationen
7  import os # Dateisystemzugriff (z.B. Prüfen ob Datei existiert)
8  import pyautogui # für die Automatisierung von GUI-Interaktionen und
   ↪ Bildschirmaufnahmen
9  # Dateiname für die gespeicherte Konfiguration
10 CONFIG_FILE = "rrt_config.json"
11
12 #=====
13 #
14 # Rapidly-exploring Random Trees (RRT)
15 # Ein moderner Ansatz zur Pfadplanung
16 # Seminararbeit im Modul Algorithmen und Datenstrukturen
17 # Autor:
18 # Wilfried Ornowski
19 # Studiengang: Informatik.Softwaresystem
20 # Semester: SoSe 2025
21 # Prüfer:
22 # Prof. Dr.-Ing. Martin Guddat
23 # Abgabedatum:
24 # 20. Mai 2025
25 # Campus Bocholt
26 # Fachbereich Wirtschaft und Informationstechnik
27 #-----
28 # RRT Algorithmus Visualisierung in 2D mit automatischen Screenshots
29 # RRT Algorithm with visual 2D rendering
30 # with automatic Screenshots
31 #=====
32
33 # Standardkonfiguration, falls keine Datei vorhanden ist
34 DEFAULT_CONFIG = {
35     "start_x": 50,
36     "start_y": 50,
37     "goal_x": 750,
38     "goal_y": 550,
39     "step_size": 20,
40     "max_iter": 1000,
41     "steps_per_frame": 1,
42     "mode": "continuous" # oder "step"
43 }
44
45 # Zeichenbereich und andere feste Werte
46 CANVAS_WIDTH = 800
```

```

47 CANVAS_HEIGHT = 600
48 NODE_RADIUS = 5
49 GOAL_RADIUS = 20
50 OBSTACLES = [
51     (200, 150, 100, 200),
52     (400, 100, 50, 300),
53     (600, 250, 150, 150)
54 ]
55
56 # Berechnet den euklidischen Abstand zwischen zwei Punkten
57 def dist(p1, p2):
58     return math.hypot(p1[0] - p2[0], p1[1] - p2[1])
59
60 # Prüft, ob die Verbindung zwischen zwei Punkten ein Hindernis durchquert
61 def is_collision(p1, p2, obstacles):
62     for (ox, oy, ow, oh) in obstacles:
63         for i in range(11): # Interpolation von 11 Zwischenpunkten
64             u = i / 10
65             x = p1[0] * (1 - u) + p2[0] * u
66             y = p1[1] * (1 - u) + p2[1] * u
67             if ox <= x <= ox + ow and oy <= y <= oy + oh:
68                 return True # Punkt liegt innerhalb eines Hindernisses
69     return False
70
71 # Gibt einen zufälligen Punkt im Zeichenbereich zurück
72 def get_random_point():
73     return random.randint(0, CANVAS_WIDTH), random.randint(0, CANVAS_HEIGHT)
74
75 # Lädt Konfigurationswerte aus Datei oder nutzt Standardwerte
76 def load_config():
77     if os.path.exists(CONFIG_FILE):
78         with open(CONFIG_FILE, "r") as f:
79             return json.load(f)
80     return DEFAULT_CONFIG.copy()
81
82 # Speichert aktuelle Konfigurationswerte in eine Datei
83 def save_config(config):
84     with open(CONFIG_FILE, "w") as f:
85         json.dump(config, f, indent=2)
86
87 # Implementiert den RRT-Algorithmus (Rapidly-exploring Random Tree)
88 class RRT:
89     def __init__(self, start, goal, obstacles, step_size):
90         self.start = start
91         self.goal = goal
92         self.obstacles = obstacles
93         self.step_size = step_size
94         self.nodes = [start] # Baumknoten
95         self.parent = {start: None} # Rückverfolgungspfad
96         self.found_path = [] # Ergebnispfad, wenn Ziel erreicht wurde
97

```

```

98     # Führt einen Schritt im RRT aus
99     def step(self):
100         rnd = get_random_point() # Zielpunkt (zufällig)
101         nearest = min(self.nodes, key=lambda p: dist(p, rnd)) # Nächstgelegener
102         ↪ Knoten
103         theta = math.atan2(rnd[1] - nearest[1], rnd[0] - nearest[0]) # Richtung
104         ↪ zum Ziel
105
106         # Erzeuge neuen Punkt in Richtung des Zielpunkts
107         new_point = (
108             int(nearest[0] + self.step_size * math.cos(theta)),
109             int(nearest[1] + self.step_size * math.sin(theta))
110         )
111
112         # Prüfe, ob der Punkt außerhalb des erlaubten Bereichs liegt
113         if not (0 <= new_point[0] <= CANVAS_WIDTH and 0 <= new_point[1] <=
114             ↪ CANVAS_HEIGHT):
115             return None, None
116
117         # Prüfe, ob die Verbindung kollidiert
118         if is_collision(nearest, new_point, self.obstacles):
119             return None, None
120
121         # Füge neuen Knoten hinzu
122         self.nodes.append(new_point)
123         self.parent[new_point] = nearest
124
125         # Zielprüfung
126         if dist(new_point, self.goal) < GOAL_RADIUS:
127             self.found_path = self.retrace_path(new_point)
128             return new_point, nearest
129
130         return new_point, nearest
131
132     # Rückverfolgungspfad vom Ziel zum Start
133     def retrace_path(self, end_point):
134         path = []
135         while end_point:
136             path.append(end_point)
137             end_point = self.parent[end_point]
138         return path[::-1] # Umkehrung für Start-Ziel
139
140     # Die GUI-Klasse zur Visualisierung und Benutzerinteraktion
141     class RRTApp:
142     def __init__(self, master):
143         self.master = master
144         self.master.title("RRT Visualisierung erweitert")
145         self.config = load_config()
146         self.entries = {} # Input-Felder
147         self.setup_ui()
148         self.running = False

```

```

146         self.iteration = 0
147
148     # Erzeugt alle GUI-Elemente
149     def setup_ui(self):
150         controls = tk.Frame(self.master)
151         controls.pack(side=tk.TOP, fill=tk.X)
152
153         # Erzeugt Eingabefelder für alle Parameter
154         for label, key in [
155             ("Start X", "start_x"), ("Start Y", "start_y"),
156             ("Goal X", "goal_x"), ("Goal Y", "goal_y"),
157             ("Step Size", "step_size"),
158             ("Max Iter", "max_iter"),
159             ("Steps / Frame", "steps_per_frame")
160         ]:
161             frame = tk.Frame(controls)
162             frame.pack(side=tk.LEFT, padx=4)
163             tk.Label(frame, text=label).pack()
164             e = tk.Entry(frame, width=6)
165             e.insert(0, str(self.config.get(key)))
166             e.pack()
167             self.entries[key] = e
168
169         # Auswahl des Ausführungsmodus
170         self.mode_var = tk.StringVar(value=self.config.get("mode",
171             ↪ "continuous"))
172         mode_menu = ttk.Combobox(controls, textvariable=self.mode_var,
173             values=["step", "continuous"],
174             ↪ state="readonly", width=10)
175         mode_menu.pack(side=tk.LEFT, padx=6)
176
177         # Buttons zum Starten und Zurücksetzen
178         tk.Button(controls, text="Start", command=self.start).pack(side=tk.LEFT,
179             ↪ padx=6)
180         tk.Button(controls, text="Reset",
181             ↪ command=self.reset_defaults).pack(side=tk.LEFT, padx=6)
182
183         # Zeichenbereich
184         self.canvas = tk.Canvas(self.master, width=CANVAS_WIDTH,
185             ↪ height=CANVAS_HEIGHT, bg="white")
186         self.canvas.pack(side=tk.LEFT)
187
188         # Seitenbereich für Koordinatenausgabe
189         side_panel = tk.Frame(self.master)
190         side_panel.pack(side=tk.RIGHT, fill=tk.Y, padx=5)
191         self.coord_text = tk.Text(side_panel, width=40, height=30)
192         self.coord_text.pack()
193         self.coord_text.config(state=tk.DISABLED)
194         self.toggle_btn = tk.Button(side_panel, text="Koordinaten ausblenden",
195             ↪ command=self.toggle_coords)
196         self.toggle_btn.pack(pady=5)

```

```

191
192     self.show_coords = True
193     self.master.bind("<space>", self.manual_step)  # Space-Taste für
        ↳ manuellen Schritt
194
195 def take_screenshot(self, reason):
196     """
197     Erstellt einen Screenshot des aktuellen Fensters.
198
199     Args:
200         reason (str): Grund für den Screenshot, wird im Dateinamen
        ↳ verwendet.
201     """
202     x = self.master.winfo_rootx()
203     y = self.master.winfo_rooty()
204     w = self.master.winfo_width()
205     h = self.master.winfo_height()
206
207     # Erstellt einen Ordner für Screenshots, falls er nicht existiert
208     os.makedirs("screenshots", exist_ok=True)
209
210     dateiname = f"screenshots/screenshot_rrt_{reason}_{self.iteration}.png"
211     screenshot = pyautogui.screenshot(region=(x, y, w, h))
212     screenshot.save(dateiname)
213     print(f"Screenshot '{dateiname}' gespeichert.")
214
215     # Koordinatenausgabe ein-/ausblenden
216 def toggle_coords(self):
217     if self.show_coords:
218         self.coord_text.pack_forget()
219         self.toggle_btn.config(text="Koordinaten einblenden")
220     else:
221         self.coord_text.pack()
222         self.toggle_btn.config(text="Koordinaten ausblenden")
223     self.show_coords = not self.show_coords
224
225     # Setzt alle Eingaben auf Standardwerte zurück
226 def reset_defaults(self):
227     for key in DEFAULT_CONFIG:
228         if key in self.entries:
229             self.entries[key].delete(0, tk.END)
230             self.entries[key].insert(0, str(DEFAULT_CONFIG[key]))
231     self.mode_var.set(DEFAULT_CONFIG["mode"])
232
233     # Startet den Algorithmus
234 def start(self):
235     self.canvas.delete("all")
236     self.iteration = 0
237     self.running = True
238
239     # Konfiguration aus Eingabefeldern lesen

```



```

240     try:
241         cfg = {key: int(entry.get()) for key, entry in self.entries.items()}
242     except ValueError:
243         messagebox.showerror("Fehler", "Bitte gültige Ganzzahlen eingeben.")
244         return
245
246     cfg["mode"] = self.mode_var.get()
247     self.config = cfg
248
249     self.start_point = (cfg["start_x"], cfg["start_y"])
250     self.goal_point = (cfg["goal_x"], cfg["goal_y"])
251
252     self.rrt = RRT(self.start_point, self.goal_point, OBSTACLES,
253                    ↪ cfg["step_size"])
254     self.max_iter = cfg["max_iter"]
255     self.steps_per_frame = cfg["steps_per_frame"]
256
257     self.draw_static()
258     if cfg["mode"] == "continuous":
259         self.master.after(10, self.update)
260
261     # Führt einen manuellen Schritt (bei Modus "step") aus
262     def manual_step(self, event=None):
263         if self.running and self.config["mode"] == "step":
264             self.update_step()
265
266     # Kontinuierlicher Ablauf (wird regelmäßig durch `after()` aufgerufen)
267     def update(self):
268         if not self.running:
269             return
270         for _ in range(self.steps_per_frame):
271             self.update_step()
272             if not self.running:
273                 break
274         if self.running and self.config["mode"] == "continuous":
275             self.master.after(5, self.update)
276
277     # Ein Einzelschritt im RRT-Ablauf
278     def update_step(self):
279         self.iteration += 1
280
281         if self.iteration in [50, 250, 500, 750, 1000]:
282             print(f"Ein Einzelschritt i= {self.iteration}") # Moderner f-String
283             ↪ für die Ausgabe
284
285         x = self.master.winfo_rootx()
286         y = self.master.winfo_rooty()
287         w = self.master.winfo_width()
288         h = self.master.winfo_height()

```

```

289         # Erstellen einen dynamischen Dateinamen
290         dateiname = f"screenshot_rrt_{self.iteration}.png"
291
292         screenshot = pyautogui.screenshot(region=(x, y, w, h))
293         screenshot.save(dateiname)
294
295         # Passe Bestätigungsmeldung dynamisch an
296         print(f"Screenshot bei Schritt {self.iteration} als '{dateiname}'
297               ↳ gespeichert.")
298
299         point, parent = self.rrt.step()
300         # self.iteration += 1
301         if point and parent:
302             self.canvas.create_line(parent[0], parent[1], point[0], point[1],
303                                   ↳ fill="blue")
304             self.show_point_info(point, parent)
305         if self.rrt.found_path:
306             self.draw_path(self.rrt.found_path)
307             # Verzögere den Screenshot um 100 Millisekunden, um sicherzustellen,
308             ↳ dass der Pfad gezeichnet ist
309             self.master.after(3000, lambda:
310                               ↳ self.take_screenshot("goal_reached") ) # Screenshot beim
311                               ↳ Erreichen des Ziels
312             self.running = False
313         elif self.iteration >= self.max_iter:
314             messagebox.showinfo("Info", "Maximale Iterationen erreicht, kein
315                                  ↳ Pfad gefunden.")
316             self.running = False
317
318         # Zeichnet statische Elemente (Hindernisse, Start/Ziel)
319         def draw_static(self):
320             for (x, y, w, h) in OBSTACLES:
321                 self.canvas.create_rectangle(x, y, x + w, y + h, fill="gray")
322             self.draw_node(self.start_point, "green", "Start")
323             self.draw_node(self.goal_point, "red", "Goal")
324
325         # Zeichnet einen Knoten (Start, Ziel oder Baumknoten)
326         def draw_node(self, pos, color, label):
327             x, y = pos
328             self.canvas.create_oval(x - NODE_RADIUS, y - NODE_RADIUS, x +
329                                   ↳ NODE_RADIUS, y + NODE_RADIUS, fill=color)
330             self.canvas.create_text(x, y - 10, text=label, fill=color)
331
332         # Zeichnet den gefundenen Pfad
333         def draw_path(self, path):
334             for i in range(len(path) - 1):
335                 self.canvas.create_line(path[i][0], path[i][1], path[i + 1][0],
336                                       ↳ path[i + 1][1], fill="orange", width=3)
337
338         # Gibt Koordinaten eines neuen Punkts im Textfeld aus
339         def show_point_info(self, current, parent):

```

```

332         if not self.show_coords:
333             return
334         self.coord_text.config(state=tk.NORMAL)
335         self.coord_text.insert(tk.END, f"Neu: {current}, Von: {parent}\n")
336         self.coord_text.see(tk.END)
337         self.coord_text.config(state=tk.DISABLED)
338
339         # Bei Beenden speichern und schließen
340         def on_close(self):
341             save_config(self.config)
342             self.master.destroy()
343
344         # Hauptprogrammstart
345         if __name__ == "__main__":
346             root = tk.Tk()
347             app = RRTApp(root)
348             root.protocol("WM_DELETE_WINDOW", app.on_close)
349             root.mainloop()

```

---

## Github 2D Quellcode

Der Quellcode und weitere Programme RRT, RRT-Connect, RRT\* mit Log File und Visualisierung befinden sich im Repository unter

[https://github.com/iaido42/RRT\\_Algorithmus](https://github.com/iaido42/RRT_Algorithmus).

## C RRT einfach erklärt

Stell dir vor, du bist in einem dunklen Labyrinth. Du kennst deinen Startpunkt und dein Ziel, aber du weißt nicht, wie die Gänge verlaufen. Also gehst du mit einer Taschenlampe los, probierst zufällig Wege aus und markierst dir die Abschnitte, die frei sind. Nach und nach entsteht ein Baum von Wegen, die alle vom Startpunkt ausgehen – und irgendwann findest du den Pfad, der dich zum Ziel bringt.

**Genau das macht der RRT-Algorithmus (Rapidly-exploring Random Tree):**

- Der Baum beginnt beim Startpunkt.
- Immer wieder wird ein zufälliger Punkt im Raum gewählt (ein sogenanntes *Sample*).
- Dann wird versucht, einen neuen Ast vom Baum in Richtung dieses Punkts wachsen zu lassen.
- Ist der Weg frei, wird der neue Punkt dem Baum hinzugefügt.

**Der Ablauf Schritt für Schritt:**

1. **Start:** Der Baum beginnt mit dem Startpunkt als Wurzel.
2. **Sampling:** Es wird ein zufälliger Punkt im Raum ausgewählt.
3. **Richtung bestimmen:** Der Punkt im Baum, der dem neuen Punkt am nächsten liegt, wird gesucht.
4. **Schrittweise Erweiterung:** Ist der neue Punkt weiter entfernt als die erlaubte Schrittweite, macht der Baum nur einen kleinen Schritt in Richtung dieses Punkts.
5. **Hindernisprüfung:** Die neue Verbindung wird auf Hindernisse überprüft.
6. **Hinzufügen:** Wenn kein Hindernis im Weg ist, wird der neue Punkt zum Baum hinzugefügt.
7. **Zieltest:** Liegt der neue Punkt nahe am Ziel? Wenn ja, ist ein Pfad gefunden.

**Wichtig:** Selbst wenn ein Sample-Punkt sehr weit weg liegt, springt der Algorithmus nicht direkt dorthin. Stattdessen macht er einen kleinen, sicheren Schritt in die Richtung – so wächst der Baum kontrolliert und kollisionsfrei.



## C.1 Änderungshistorie

### Änderungshistorie


Version	Änderungen
20. Mai 2025	<p><b>Erste Abgabe</b> der Seminararbeit mit dem Titel <i>Rapidly-exploring Random Trees (RRT) – Ein moderner Ansatz zur Pfadplanung</i>. Beinhaltet:</p> <ul style="list-style-type: none"><li>• Einführung in RRT und verwandte Pfadplanungsalgorithmen</li><li>• Beschreibung des Basisalgorithmus inkl. Pseudocode und Visualisierung</li><li>• Varianten wie RRT*, RRT-Connect, Informed RRT</li><li>• Erste Benchmarks und Beispielimplementierungen in Python, Godot (GDScript), C++</li></ul>
9. Juni 2025 (v13)	<p><b>Überarbeitete Version</b> mit folgenden Änderungen:</p> <ul style="list-style-type: none"><li>• Überarbeiteter Pseudocode in Abschnitt 3.1 für bessere Lesbarkeit und Struktur</li><li>• Erweiterung der Tabelle 3 mit Symbolen zur besseren Unterscheidung optimaler Varianten</li><li>• Korrektur stilistischer und sprachlicher Formulierungen in Einleitung und Zielsetzung</li><li>• Ergänzungen im Abschnitt zu <i>Informed RRT*</i> mit genauerer Beschreibung der Ellipsoid-Sampling-Technik</li><li>• Einheitliche Zitierweise und Korrektur kleiner typografischer Fehler</li><li>• <b>Entfernung der redundanten Tabelle 5</b> (Duplikat von Tabelle 4) zur Klassifikation von Pfadplanungsalgorithmen</li></ul>
9. Juni 2025 (v14)	<p><b>Feinschliff und Erweiterung</b> mit folgenden Änderungen:</p> <ul style="list-style-type: none"><li>• Einleitung positiver formuliert</li><li>• Repository mit RRT, RRT-Connect, RRT* erweitert</li><li>• Anhang B "RRT Varianten hinzugefügt"</li><li>• Anhang C "RRT einfach erklärt" hinzugefügt</li><li>• Querverweise, Glossar aus dem Text zum neuen Anhang ergänzt</li><li>• Korrekturen bei Literaturverweisen und Fußnotenformatierung</li></ul>



## C.2 Selbstständigkeitserklärung

Ich erkläre, dass ich alle Quellen, die ich zur Erstellung dieser Arbeit verwendet habe, im Quellenverzeichnis aufgeführt habe und dass ich alle Stellen, an denen Informationen (Texte, Bilder) aus diesen Quellen in meine Arbeit eingeflossen sind, als Zitate mit Quellenangabe kenntlich gemacht habe.

Mir ist bewusst, dass ein Verstoß gegen diese Regeln zum Ausschluss aus dem Seminar führt.

A handwritten signature in black ink, appearing to read 'Wilfried Ornowski', written in a cursive style.

---

Unterschrift

**WILFRIED ORNOWSKI**