Karlsruhe Institute of Technology

Innovative Konzepte zur Programmierung von Industrierobotern

1st August 2025

Projektaufgabe - Hierachischer Planer Umsetzung und Implementierung

Simon Boes und Mert Baran Unor

Project

Optional additional info

Contents

Chapter 1
Einleitung

Im Rahmen der Projektaufgabe im Kurs *Innovative Konzepte zur Programmierung von Industrieroботern* bestand die Zielsetzung darin, ein **hierarchisches Bewegungsplanungsverfahren** auf Basis probabilistischer Roadmaps (PRM) zu implementieren und zu evaluieren. Hierbei sollte eine zweistufige Struktur aufgebaut werden, bei der ein globaler Planer (Hauptplaner) grob den Weg zwischen Start- und Zielkonfiguration vorgibt, während ein lokaler Planer (Subplaner) für die Feinkollisionserkennung und Pfadausarbeitung zwischen diesen Segmenten zuständig ist.

Die Aufgabe beinhaltete neben der Implementierung auch die Erstellung von Testszenarien für Roboter mit verschiedenen Freiheitsgraden (2, 4, 8, 12 DoF) sowie die Evaluation mehrerer PRM-Varianten (Standard PRM, Lazy PRM, Visibility-Roadmap) im Vergleich. Besonderes Augenmerk lag auf der Fähigkeit des hierarchischen Ansatzes, komplexe Bewegungsplanungsprobleme modular, robust und effizient zu lösen.

Der Schwerpunkt dieses Berichts liegt auf der Umsetzung und dem Vergleich zweier unterschiedlicher Ansätze zur hierarchischen PRM-Planung: einer ersten, konzeptionell fehlerhaften Variante sowie einer überarbeiteten Implementierung, bei der die interne Planung die klassische Sichtbarkeitsprüfung ersetzt. Beide Varianten werden analysiert, diskutiert und auf ihre Stärken und Schwächen hin untersucht.

Chapter 2

Umsetzung

2.1 Erste Implementierung (fehlerhafter Ansatz)

Die erste Version des hierarchischen PRM-Planers war zwar prinzipiell lauffähig, wich jedoch in wesentlichen Punkten von der intendierten Architektur ab. Ziel war es ursprünglich, dass ein globaler Hauptplaner eine grobe Trajektorie erzeugt, die anschließend durch einen lokalen Subplaner segmentweise verfeinert wird. In der ersten Umsetzung war diese Trennung allerdings nicht konsequent realisiert.

2.1.1 Idee hinter dem Ansatz

Die Grundidee bestand darin, den durch den Hauptplaner (*Visibility PRM*) gefundenen Lösungspfad als Grundlage für eine nachgelagerte Subplanung zu verwenden. Konkret sollte der Lösungspfad segmentweise an einen zweiten Planer (LazyPRM oder BasicPRM) übergeben werden, um so eine robustere Lösung zu generieren.

Der vermutete Vorteil dieses Verfahrens lag darin, dass die Anzahl der Knoten im Subplaner drastisch reduziert werden konnte. Anstatt eine komplette Roadmap im gesamten Konfigurationsraum aufzubauen, wurde der Suchraum auf die Umgebung des bereits gefundenen Lösungspfads beschränkt. Dies versprach eine effizientere Nutzung von Ressourcen und eine gewisse Robustheit gegenüber Änderungen in der Umgebung. Selbst wenn der Visibility-Planer in einer veränderten Szene keine direkte Verbindung mehr erkennen würde, könnte ein lokal gestreuter Lazy- oder Basic-PRM alternative Pfade erzeugen.

2.1.2 Praktische Umsetzung und Problemstellung

Das Verfahren wurde jedoch fehlerhaft interpretiert: Anstelle den einfachen Linientest im Hauptplaner durch eine Subplanung zu ersetzen, wurden zwei vollständige Algorithmendurchläufe durchgeführt. Zunächst erfolgte die Pfadberechnung durch den Visibility PRM, und erst anschließend wurde auf diesem Pfad eine zusätzliche Subplanung gestartet. Damit wurde das intendierte hierarchische Prinzip verfehlt, da der eigentliche Linientest unverändert blieb und die Subplanung lediglich nachgeschaltet wurde.

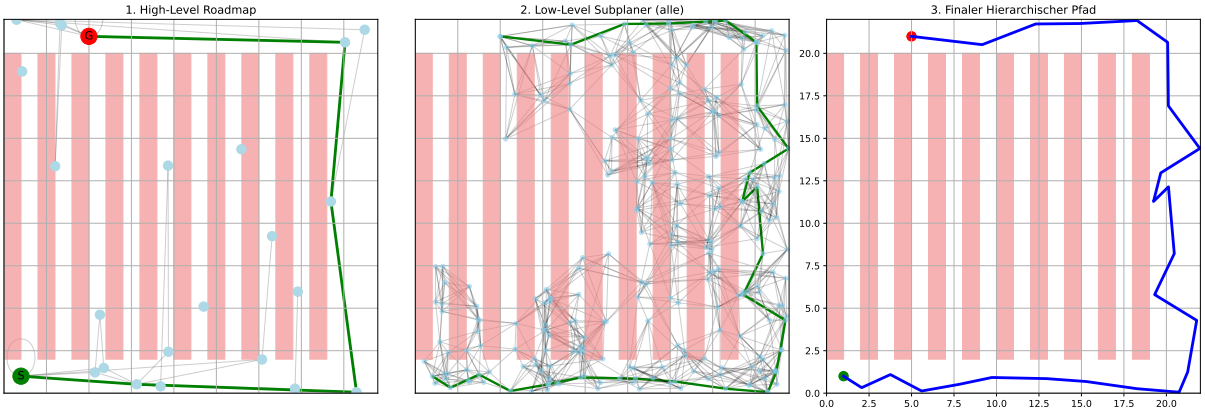


Figure 2.1 Visualisierung der ersten Implementierung: (links) High-Level Roadmap durch Visibility PRM, (mitte) nachgeschaltete Lazy PRM Subplaner-Auswertung entlang der Segmente, (rechts) resultierender kombinierter Pfad.

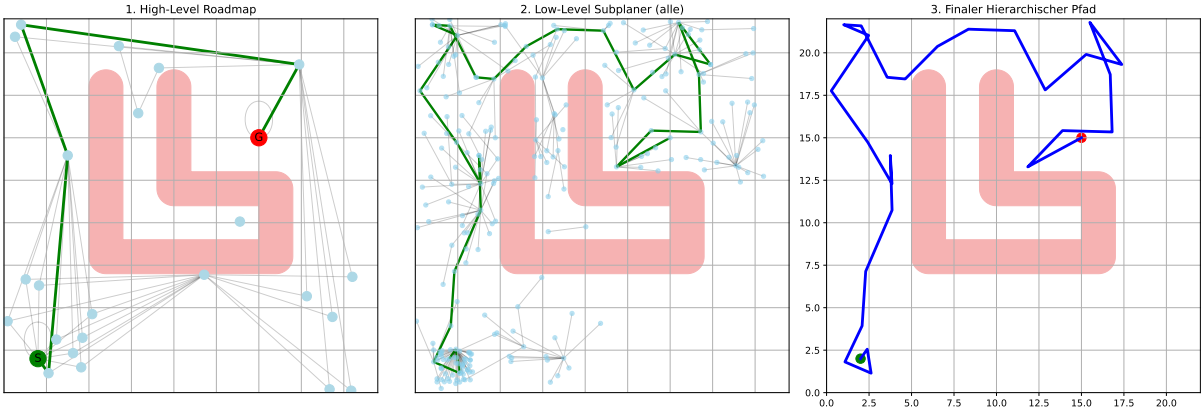


Figure 2.2 Visualisierung der ersten Implementierung: (links) High-Level Roadmap durch Visibility PRM, (mitte) nachgeschaltete BasicPRM Subplaner-Auswertung entlang der Segmente, (rechts) resultierender kombinierter Pfad.

2.1.3 Zusammenfassung der Probleme

Die erste Version des Planers litt daher unter folgenden Kernproblemen:

- **Fehlerhafte Interpretation der Aufgabenstellung:** Der Linientest wurde nicht ersetzt, sondern lediglich ein zweiter Planungsdurchlauf angehängt.
- **Keine tatsächliche Hierarchie:** Der Subplaner war funktional überflüssig, da der Hauptplaner bereits vollständige Pfade erzeugte.
- **Ineffizienz:** Zwei komplette Algorithmendurchläufe führten zu erhöhtem Rechenaufwand.
- **Fehlende Modularität:** Komponenten wie Start-/Zielverbindung, Sichtbarkeitsgraph und Subplanung waren nicht klar voneinander getrennt.

Trotz dieser Probleme zeigte sich die Idee, einen Lösungspfad durch zusätzliche Subplanung abzusichern, als interessanter Ansatz für robustere Roadmaps. Diese Überlegung diente als Motivation für die Entwicklung der finalen Implementierung.

2.2 Hybrider Ansatz (Abwandlung der ersten Implementierung)

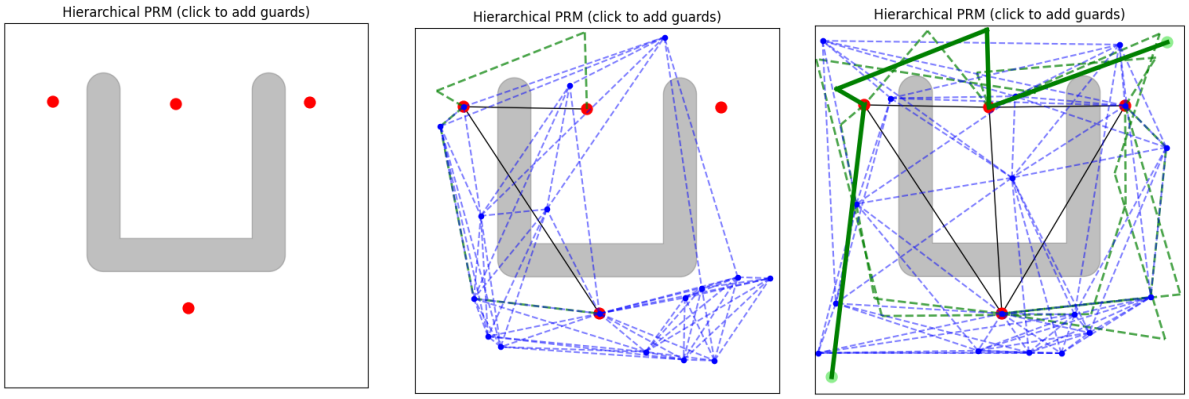
Aufbauend auf den Erfahrungen mit der ersten Implementierung wurde ein Zwischenschritt entwickelt, der die Grundidee des Visibility-Planers beibehielt, jedoch in modifizierter Form umgesetzt wurde. Dieser Ansatz diente als **schnelle Anpassung**, um die Architektur näher an die intendierte Aufgabenstellung heranzuführen, ohne den Linientest bereits vollständig zu ersetzen.

2.2.1 Konzept

Die zentrale Idee bestand darin, zunächst ausschließlich die sogenannten *Guards* im Konfigurationsraum zu platzieren. Dies erfolgte weiterhin durch eine Sichtprüfung (Linientest), sodass die Gesamtanzahl der Knoten auf ein Minimum reduziert wurde. Nach Abschluss der Guard-Platzierung wurden die Guards paarweise betrachtet, und für jede potenzielle Verbindung kam ein Subplaner (LazyPRM oder BasicPRM) zum Einsatz.

Auf diese Weise entstand eine Roadmap, die im Wesentlichen aus einer minimalen Menge von Guards besteht, während die eigentlichen Verbindungen zwischen ihnen durch Subplanung erzeugt werden. Gegenüber dem ersten Ansatz, bei dem zwei vollständige Algorithmendurchläufe durchgeführt wurden, stellt dies eine deutliche Verbesserung dar.

Zur Umsetzung wurde eine einfache Visualisierung mit `Matplotlib` entwickelt. Sie erlaubte die manuelle Platzierung der Guards sowie die schrittweise Ausführung des Algorithmus, bis hin zur Ermittlung eines Pfades von Start zu Ziel. Eine entsprechende Darstellung ist in Abbildung 2.3 gezeigt. Die Inspiration zu dieser Visualisierung entstand durch die Präsentation einer weiteren Projektgruppe, die ein ähnliches Vorgehen für ihre Problemstellung gewählt hatte.



(a) Platzierung der Guards **(b)** Verbindungen durch Subplaner **(c)** Resultierender Pfad

Figure 2.3 Illustration des hybriden Ansatzes: (a) Platzierung der Guards, (b) Berechnung der Verbindungen durch Subplaner, (c) finaler kombinierter Pfad von Start zu Ziel.

Abbildung 2.3c zeigt den finalen Pfad als durchgezogene grüne Linie. Die grün gestrichelten Linien repräsentieren die durch den Subplaner gefundenen Verbindungen zwischen den in Abbildung 2.3a platzierten Guards.

2.2.2 Bewertung

Dieser Ansatz kann als **hybride Variante** eingeordnet werden:

- **Positiv:** Die Anzahl der Knoten wird stark reduziert, und die Subplaner sorgen für robustere Verbindungen.
- **Negativ:** Der Linientest wurde nicht vollständig ersetzt, sondern lediglich für die Verbindungserzeugung ergänzt.

Damit erfüllt das Verfahren die Idee eines hierarchischen Planers nur teilweise. Es stellt jedoch ein **Übergangsmodell** dar, das eine Brücke zwischen der fehlerhaften ersten Implementierung und der finalen, vollständig hierarchischen Lösung bildet.

2.3 Finale Implementierung (vollständig hierarchischer Ansatz)

Nachdem der hybride Ansatz noch nicht die Anforderungen der Projektaufgabe erfüllte, wurde eine dritte Implementierung entwickelt, in der der Linientest im Hauptplaner **konsequent durch einen Subplaner ersetzt** wurde. Dieses Verfahren entspricht damit dem in der Aufgabenstellung geforderten hierarchischen Planungsansatz.

2.3.1 Konzept

Der Hauptplaner basiert weiterhin auf einer Guard-basierten Visibility-Roadmap. Die wesentliche Änderung besteht jedoch darin, dass Sichtbarkeitsprüfungen (`Linientest`) nicht mehr direkt durchgeführt werden, sondern für jede potenzielle Verbindung ein **lokaler PRM** innerhalb einer begrenzten Region of Interest (ROI) erzeugt wird.

- Die ROI wird als rechteckiger Bereich um die beiden zu verbindenden Guards definiert, erweitert um einen konfigurierbaren Sicherheitsabstand.
- Innerhalb dieser ROI wird ein neuer `LazyPRM` oder `BasicPRM` instanziiert, der ausschließlich für diese Verbindung einen Pfad plant.
- Gelingt die Subplanung, so wird die Verbindung in den High-Level-Graphen aufgenommen. Andernfalls bleibt die Verbindung unberücksichtigt.

Damit wird jede Kante im globalen Graphen explizit durch eine lokale Subplanung abgesichert. Der Hauptplaner liefert lediglich die Struktur (Guards und Kandidaten für Verbindungen), während die eigentliche Validierung und Pfaderzeugung vollständig auf die Subplaner ausgelagert ist.

2.3.2 Algorithmische Umsetzung

Die Implementierung (vgl. Codeauszug in Anhang X) folgt diesem Ablauf:

1. Guards werden iterativ im Konfigurationsraum platziert, wobei nur Punkte ohne direkte Sichtlinie zu bestehenden Guards akzeptiert werden.
2. Für jede potenzielle Verbindung zwischen einem neuen Punkt und bestehenden Guards wird eine ROI bestimmt und ein lokaler Subplaner gestartet.
3. Der Subplaner liefert entweder einen kollisionsfreien Pfad oder schlägt fehl. Nur im Erfolgsfall wird eine gewichtete Kante (nach euklidischer Distanz) im High-Level-Graphen hinzugefügt.
4. Start- und Zielkonfiguration werden analog über lokale Subplanung an den High-Level-Graphen angebunden.
5. Die finale Pfadberechnung erfolgt mittels `shortest_path` auf dem gewichteten High-Level-Graphen.

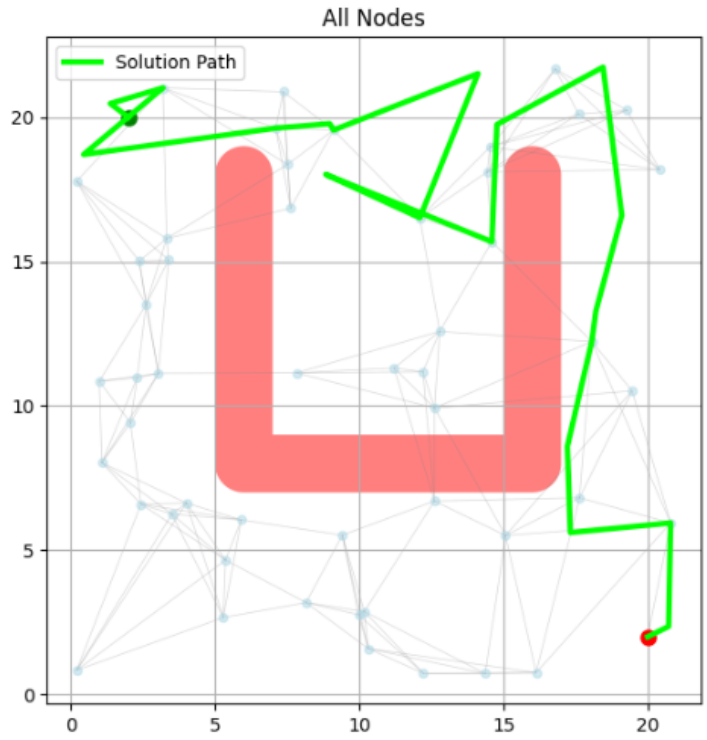


Figure 2.4 Darstellung der finalen Implementierung: Der resultierende Pfad (grün) wird durch den Hauptplaner vorgegeben und in allen Verbindungen konsequent durch lokale Subplaner abgesichert. Die roten Flächen markieren Hindernisse, während die blauen Punkte die Knoten im High-Level-Graphen darstellen.

2.3.3 Bewertung

Dieses Verfahren stellt eine **vollständig hierarchische Lösung** dar:

- Der Hauptplaner (Visibility-Roadmap) liefert nur die abstrakte Struktur.
- Der Subplaner übernimmt sämtliche Verbindungsprüfungen und garantiert Kollisionsfreiheit.
- Jeder Subplaner arbeitet innerhalb einer lokal begrenzten ROI, wodurch die Planungsaufgabe effizient und modular bleibt.

Damit erfüllt die finale Implementierung exakt die in der Aufgabenstellung formulierten Anforderungen an einen hierarchischen Bewegungsplaner. Sie ist modular aufgebaut, robust gegenüber fehlerhaften Verbindungen und erlaubt eine klare Trennung zwischen globaler Struktur und lokaler Pfaderzeugung.

Chapter 3
Evaluation und Tests

3.1 Anpassung der Testumgebung

Für die Evaluation der verschiedenen Implementierungen wurde der bestehende `PlanarRobot`-Code erweitert. Durch die Parametrisierung über die Anzahl der Gelenke (`n_joints`) und die Gesamtlänge des Arms (`total_length`) konnte für jede Testumgebung ein konsistentes Robotermodell erzeugt werden.

Die Start- und Zielkonfigurationen wurden dabei mit der Hilfsfunktion `generate_consistent_joint_config` erzeugt, sodass stets eine geometrisch vergleichbare Pose vorlag. Dies garantierte, dass sowohl einfache als auch komplexere Roboter (z. B. 2-DoF, 4-DoF, 6-DoF oder 12-DoF) unter gleichen Bedingungen getestet werden konnten.

3.1.1 Beispiel: Erstellung eines 12-DoF Roboters

Zur Demonstration der Skalierbarkeit wurde ein Planarroboter mit zwölf Gelenken erzeugt. Die Parametrisierung erfolgt dabei über die Anzahl der Gelenke (`n_joints`) sowie die Gesamtlänge des Arms (`total_length`). Zusätzlich werden die Bewegungsgrenzen jedes Gelenks über das `limits`-Array definiert. Die Start- und Zielkonfigurationen werden mit der Hilfsfunktion `generate_consistent_joint_config` generiert, wodurch eine konsistente Geometrie sichergestellt ist.

Listing 3.1 Erzeugung eines 12-DoF Planarroboters mit Start- und Zielkonfiguration

```
r = PlanarRobot(n_joints=12, total_length=3.5)
limits = [[-3.14,3.14]] * r.dim
environment = KinChainCollisionChecker(r, obst, limits=limits, fk_resolution=.2)

start_joint_angles = generate_consistent_joint_config(r.dim, total_angle=2, curvature=0.45)
start_joint_pos = sp.Matrix(start_joint_angles)

end_joint_angles = generate_consistent_joint_config(r.dim, total_angle=-1.85, curvature=-0.45)
end_joint_pos = sp.Matrix(end_joint_angles)

fig_local = plt.figure(figsize=(14, 7))
ax1 = fig_local.add_subplot(1, 2, 1)
ax1.set_xlim([-4,4])
ax1.set_ylim([-4,4])
environment.drawObstacles(ax1, True)
r.move(start_joint_pos)
planarRobotVisualize(r, ax1)
r.move(end_joint_pos)
planarRobotVisualize(r, ax1)
```

Mit diesem Vorgehen konnte der Roboter sowohl in der Start- als auch in der Zielkonfiguration visualisiert und für die anschließende Pfadplanung verwendet werden. Abbildung 3.1 zeigt exemplarisch die Darstellung eines niedrigdimensionalen Falls (2-DoF); das Prinzip lässt sich jedoch direkt auf höhere Dimensionen übertragen.

3.2 Durchführung der Tests

Die Tests erfolgten schrittweise für Roboter mit unterschiedlicher Gelenkzahl:

- **2-DoF:** Einfache kinematische Kette mit zwei Freiheitsgraden, geeignet zur Visualisierung des Basisverhaltens der Planer.
- **4-DoF:** Mittlere Komplexität, erste signifikante Zunahme der Konfigurationsraumdimension.
- **6-DoF:** Vergleichbar mit klassischen Industrierobotern, relevante Testgröße für praktische Szenarien.

- **12-DoF:** Sehr hohe Komplexität, zur Untersuchung der Skalierbarkeit und Robustheit der Verfahren.

Für jede dieser Konfigurationen wurden die Planungsverfahren (`LazyPRM`, `BasicPRM`, `VisibilityPRM` sowie die hierarchische Variante) auf denselben Start-/Zielzustand angewandt. Dadurch konnten die Ergebnisse hinsichtlich Pfadlänge, Berechnungszeit und Stabilität direkt verglichen werden.

3.3 Visualisierung

Zur Überprüfung der Ergebnisse wurden die Testumgebungen in `Matplotlib` dargestellt. Die Hindernisse wurden durch Polygone oder Liniensegmente mit Pufferung modelliert, sodass ein realistischer Kollisionsraum entstand. Die Pfade wurden anschließend animiert, um den Bewegungsablauf der verschiedenen Roboterkonfigurationen anschaulich darzustellen.

Ein Beispiel einer 2-DoF-Szene ist in Abbildung 3.1 gezeigt.

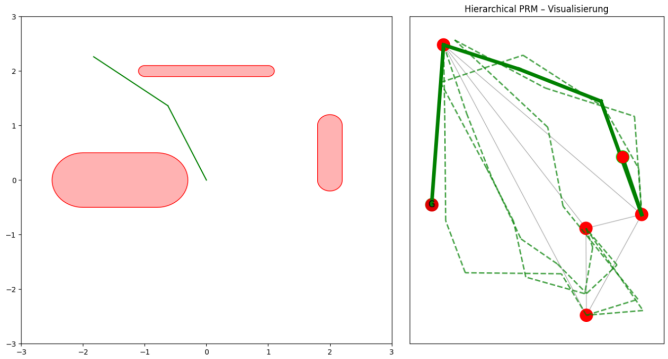


Figure 3.1 Beispielhafte Visualisierung einer 2-DoF-Testumgebung mit Start- und Zielkonfiguration. Hindernisse sind in Rot dargestellt, der Roboterarm in den jeweiligen Posen in Schwarz.

3.4 Beobachtungen und Ergebnisse

3.4.1 2-DoF

Bei einem Roboter mit zwei Freiheitsgraden konnten alle getesteten Verfahren (`LazyPRM`, `BasicPRM`, `VisibilityPRM` sowie der hierarchische Ansatz) stabile und kollisionsfreie Pfade berechnen. Die Berechnungszeiten lagen im niedrigen Sekundenbereich, sodass die Ergebnisse sowohl für Animationszwecke als auch für statistische Auswertungen gut geeignet waren. In dieser geringen Dimension zeigten sich kaum Unterschiede in der Leistungsfähigkeit.

3.4.2 4-DoF

Bereits bei vier Freiheitsgraden stieg der Rechenaufwand signifikant an. Während der `LazyPRM` noch akzeptable Berechnungszeiten lieferte, verlängerten sich die Laufzeiten beim `VisibilityPRM` bereits deutlich. Insbesondere die Standardvariante des `VisibilityPRM` zeigte, dass mit zunehmender Dimensionalität die Roadmap komplexer wird, was zu längeren Laufzeiten und teilweise instabilen Ergebnissen führte. Auch der hierarchische Ansatz war in dieser Dimension zwar lauffähig, benötigte aber erheblich mehr Zeit pro Berechnung als bei 2-DoF.

3.4.3 6-DoF

Bei sechs Freiheitsgraden konnten mit dem `LazyPRM` noch Lösungen berechnet werden, allerdings mit deutlich höherem Zeitaufwand (teils mehrere Minuten pro Pfad). Beim hierarchischen Ansatz verlängerte sich die Rechenzeit jedoch dramatisch: Einzelne Planungen dauerten über zwei Stunden, ohne dass zuverlässig ein Ergebnis erzielt wurde. Dies deutet darauf hin, dass entweder

- noch ein Implementierungsfehler im hierarchischen Planer vorliegt (z. B. unnötig mehrfach ausgeführte Berechnungen),
- oder die gewählten Parameter (Region of Interest, Knotenzahl pro Subplaner, Suchradius) nicht optimal gewählt sind und angepasst werden müssten.

Aufgrund der begrenzten Projektzeit konnten diese Parameter jedoch nicht weiter optimiert werden. Daher blieb offen, ob die hohen Laufzeiten allein auf eine ineffiziente Parametrisierung zurückzuführen sind oder tatsächlich ein Fehler in der Implementierung vorliegt.

3.4.4 12-DoF

Für zwölf Freiheitsgrade war es im gegebenen Zeitrahmen nicht möglich, konsistente Lösungen mit allen Planungsverfahren zu berechnen. Insbesondere der `VisibilityPRM` und der hierarchische Ansatz waren hier aufgrund der exponentiell steigenden Rechenzeit nicht mehr durchführbar. Der `LazyPRM` zeigte sich prinzipiell robuster, konnte aber ebenfalls keine Ergebnisse in angemessener Zeit liefern.

3.5 Zusammenfassung

Insgesamt gilt:

- **Bis 2-DoF:** Alle Verfahren zuverlässig und schnell.
- **Bis 4-DoF:** LazyPRM performant, VisibilityPRM und hierarchischer Ansatz bereits deutlich langsamer.
- **Bis 6-DoF:** LazyPRM liefert noch Ergebnisse, allerdings mit sehr langen Laufzeiten. Der hierarchische Ansatz benötigte über zwei Stunden und deutet auf Optimierungsbedarf oder Implementierungsfehler hin.
- **Ab 12-DoF:** Keine praktikablen Ergebnisse in der verfügbaren Zeit.

Durch die verspätete funktionierende Implementierung (bedingt durch den fehlerhaften ersten Ansatz) stand nur eine begrenzte Zeit für die umfangreiche Testphase zur Verfügung. Daher konnte keine vollständige Benchmarking-Serie über alle Freiheitsgrade und Planer durchgeführt werden. Die vorliegenden Ergebnisse geben jedoch einen klaren Hinweis darauf, dass insbesondere der hierarchische Ansatz in höheren Dimensionen noch erheblich optimiert werden müsste, um konkurrenzfähig zu sein.

Chapter 4

Beantwortung der Fragen im Endbericht

4.1 Single-Query- und Multi-Query-Verfahren

4.1.1 Definitionen

- **Single-Query-Verfahren:** Beantworten genau einer Anfrage von Start- zu Zielzustand, danach Verwerfen der aufgebauten Datenstruktur.
- **Multi-Query-Verfahren:** Aufbau eines Konfigurationsraum-Graphen (Roadmap), der für viele Anfragen wiederverwendet wird.

4.1.2 Beispiele und Einordnung

RRT & RRT-Connect typische Single-Query-Algorithmen, gut geeignet für dynamische Umgebungen [3].

PRM & PRM* klassische Multi-Query-Verfahren mit teurem Preprocessing und schneller Pfadsuche mittels Graph-Algorithmen [2].

4.2 Bewegungsbefehle für die Pfadauflösung

Gegeben einen Pfad aus Konfigurationen

$$\{q_0, q_1, \ldots, q_n\},$$

kann man in Robotersprachen z. B. folgende Befehle nutzen:

Steuerung	Befehl	Beschreibung
ABB (RAPID)	`MoveJ(q_i)`	Gelenkraum-Punkt-zu-Punkt
ABB (RAPID)	`MoveL(p)`	Lineare Bahn im Kartesischen Raum
KUKA (KRL)	`PTP q_i`	Punkt-zu-Punkt im Gelenkraum
KUKA (KRL)	`LIN p`	Lineare Interpolation im Kartesischen Raum

Table 4.1 Beispiele für Bewegungsbefehle

Warum hier nicht ideal?

- Abrupte Richtungswechsel zwischen diskreten Konfigurationen.
- Ungeglättete Trajektorien führen zu Ruckeln.
- Große Anzahl sequentieller Befehle erschwert Wartung.

4.3 Voraussetzungen für schnelles Abfahren

1. **Zeitparameterisierung:** Kontinuierliches Geschwindigkeitsprofil entlang des Pfades [3].
2. **Pfadsmoothing:** Shortcutting oder Spline-basierte Glättung, um Krümmungssprünge zu eliminieren.
3. **Bahnfolgeregelung:** Einsatz eines Regelkreises zur Kompensation von Abweichungen.
4. **Begrenzung von Geschwindigkeit und Beschleunigung:** Schutz der Mechanik und Einhaltung sicherheitstechnischer Vorgaben.

4.4 Verbesserung der Pfadqualität

1. **Asymptotisch optimale Algorithmen:** RRT* und PRM* für Pfade, die mit steigender Rechenzeit optimaler werden [1].
2. **Erweitertes Sampling:** Goal- und Obstacle-Biasing zur besseren Abdeckung relevanter Regionen [1].
3. **Post-Processing:** Shortcutting zur Verkürzung der Pfadlänge.
4. **Re-Planning:** Dynamische Anpassung bei Änderungen der Umgebung.

Chapter 5
Diskussion

Die durchgeführten Arbeiten haben gezeigt, dass die Entwicklung eines hierarchischen Planungsverfahrens für Industrieroboter eine komplexe und fehleranfällige Aufgabe darstellt. Besonders deutlich wurde dies durch den anfänglich fehlerhaften Ansatz, bei dem die Trennung zwischen Haupt- und Subplaner nicht konsequent umgesetzt war. Die daraus resultierenden Schwierigkeiten führten zwar zu einem erheblichen Zeitverlust, ermöglichten jedoch auch eine bewusste Auseinandersetzung mit den konzeptionellen Anforderungen.

Der hybride Ansatz stellte einen wichtigen Zwischenschritt dar, da er die Vorteile einer Guard-basierten Struktur mit Subplanung für Kanten verband. Auch wenn er nicht der exakten Aufgabenstellung entsprach, konnte dadurch eine Brücke zwischen dem fehlerhaften Start und der finalen Implementierung geschaffen werden. Die finale Version zeigte schließlich das intendierte hierarchische Verhalten, indem jede Verbindung im High-Level-Graphen durch eine lokale Subplanung validiert wurde.

Die Tests verdeutlichten jedoch auch die Grenzen des Ansatzes. Während bei niedrigen Freiheitsgraden (2-DoF) alle Verfahren performant und stabil waren, stieg der Rechenaufwand mit zunehmender Dimensionalität drastisch an. Besonders der hierarchische Planer zeigte bei 6-DoF Laufzeiten von über zwei Stunden, was auf nicht optimale Parameterwahl (Region of Interest, Knotenzahl) oder mögliche Implementierungsfehler hindeutet. Aufgrund des begrenzten Zeitrahmens war es nicht möglich, diese Ursachen weitergehend zu analysieren oder eine systematische Parametertuning-Studie durchzuführen.

Zusammenfassend kann festgehalten werden, dass die konzeptionelle Idee des hierarchischen Planers erfolgreich umgesetzt wurde, die praktische Leistungsfähigkeit in höheren Dimensionen jedoch noch deutlich eingeschränkt ist.

Chapter 6

Fazit

Im Rahmen dieses Projekts wurde ein hierarchischer Bewegungsplaner für Industrieroboter entworfen, implementiert und evaluiert. Die Arbeit verlief in drei Schritten: (1) ein fehlerhafter erster Ansatz ohne klare Hierarchie, (2) ein hybrider Übergangsansatz, und (3) eine finale Implementierung mit vollständiger Ersetzung des Linientests durch lokale Subplaner in einer Region of Interest.

Die Tests zeigten, dass das Verfahren in niedrigen Dimensionen (2-DoF) zuverlässig funktioniert, in höheren Dimensionen jedoch erhebliche Rechenzeiten auftreten. Dies weist darauf hin, dass der Ansatz zwar prinzipiell geeignet ist, aber noch nicht in der aktuellen Form für komplexe industrielle Anwendungen taugt.

Für eine zukünftige Weiterentwicklung ergeben sich daher folgende Punkte:

- **Parametertuning:** Anpassung der ROI-Größe, Knotenzahl und Suchparameter zur Reduktion der Rechenzeiten.
- **Optimierung der Implementierung:** Analyse möglicher redundanter Berechnungen und algorithmische Verbesserungen der Subplaner.
- **Erweiterte Tests:** Durchführung systematischer Benchmark-Studien über höhere DoF und verschiedene Umgebungen.
- **Praktische Integration:** Übertragung des Konzepts auf realistische Roboterkinematiken und industrielle Szenarien.

Trotz der aufgetretenen Schwierigkeiten konnte das Projekt erfolgreich zeigen, wie ein hierarchischer Planer konzipiert und umgesetzt werden kann. Die Arbeit liefert damit eine wertvolle Grundlage für weiterführende Forschung und Entwicklung im Bereich innovativer Konzepte zur Programmierung von Industrierobotern.

Chapter 7

Fazit

Die Projektarbeit führte trotz anfänglicher Schwierigkeiten zu einer erfolgreichen Umsetzung eines robusten und modularen hierarchischen Planers. Die entwickelte Methode zeichnet sich besonders durch ihre Flexibilität, Fehlertoleranz und Robustheit gegenüber dynamischen Veränderungen aus. Zukünftige Arbeiten könnten insbesondere auf eine Verbesserung der Skalierbarkeit für höhere Freiheitsgrade sowie auf eine Integration in realitätsnahe Robotersteuerungen und Visualisierungssysteme abzielen.

Bibliography

[1] Sertac Karaman and Emilio Frazzoli. Sampling-based algorithms for optimal motion planning. *The International Journal of Robotics Research*, 30(7):846–894, 2011.

[2] Lydia E. Kavraki, Pavel Švestka, Jean-Claude Latombe, and Mark H. Overmars. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *IEEE Transactions on Robotics and Automation*, 12(4):566–580, 1996.

[3] Steven M. LaValle. *Planning Algorithms*. Cambridge University Press, 2006.