Studienarbeit – Mähroboter Wegsteuerung

Inhaltsverzeichnis

1. Einleitung

Diese Studienarbeit wurde im Rahmen des Moduls Software Engineering (KI-B-4) im Studiengang Künstliche Intelligenz erstellt. Ziel war die Entwicklung eines Softwaremoduls zur Wegsteuerung eines simulierten Mähroboters. Die Arbeit wurde im Rahmen eines agilen Prozesses iterativ realisiert. Wert wurde dabei auf Modularität, Testbarkeit und eine durchgehende Traceability gelegt.

2. Requirements Engineering

2.1 Stakeholder-Analyse

• Endanwender: Personen, die einen automatisierten Mähroboter nutzen möchten.

• Entwickler: Tobias Litke

• Prüfer: Prof. Dr. Karsten Becker

2.2 Anforderungen

Funktionale Anforderungen:

1) Navigation zum Zielpunkt
2) Hindernisvermeidung durch Pfadberechnung
3) ASCII-Visualisierung der Umgebung
4) Ausgabe von Routendaten
5) Steuerung mehrerer Levels mit Statistik

Nicht-funktionale Anforderungen:

1) Modularer Aufbau
2) Rückverfolgbarkeit
3) Plattformunabhängigkeit (Java, Konsole)

2.3 User Stories

• Als Nutzer möchte ich, dass der Roboter ein Ziel ansteuert.

• Als Entwickler möchte ich sicherstellen, dass Hindernisse korrekt erkannt werden.

• Als Prüfer möchte ich Iterationen mit überprüfbarem Fortschritt sehen.

2.4 Priorisierung und Iterationen

Die Anforderungen wurden nach MoSCoW-Prinzip priorisiert:

- **Must**: Navigation, Hindernisse, Zielerreichung

- **Should**: Statistik, Wiederholung

- **Could**: grafische Oberfläche

3. Architektur & Design

Die Architektur basiert auf fünf klar getrennten Modulen: Point, GoalManager, Map, Logger, SimulationEngine.

Diese sind lose gekoppelt und kommunizieren über wohldefinierte Schnittstellen.

Ein UML-Klassendiagramm dokumentiert die Beziehungen visuell (siehe Anhang).

4. Iterationen

1) **Navigation zum Zielpunkt (ohne Hindernisse)**

   - Aufbau der Navigationslogik
   - Verarbeitung einfacher Wegpunktlisten
   - Integration eines Sensor-Simulators

2) **Pfadberechnung mit Hindernissen, ASCII-Ausgabe**

   - Berechnung alternativer Wege bei Hindernisdetektion
   - Erste Tests mit statischen Hindernissen
   - Verbesserte Zielwahl-Strategie

3) **Levelsteuerung, Statistik, Logger**

   - Implementierung eines einfachen Simulationsinterfaces (Textbasiert)
   - Logging der Fahrtroute
   - Konsistente Fehlerbehandlung und Validierung

5. Test und Qualitätssicherung

5.1 Traceability Matrix

Requirement	Design-Komponente	Implementierung	Testfall
Zielnavigation	GoalManager	GoalManager.java	testPathFound()
Hindernisvermeidung	GoalManager	GoalManager.java	testNoPathWhenBlocked()
Objektgleichheit	Point	Point.java	PointTest
Konsolenausgabe	Map, Logger	Map.java, Logger.java	manuell

5.2 Teststrategie

- Unit-Tests für Navigations- und Zielsteuerungskomponenten
- Integrationstests der Gesamtlogik

- Simulierte Testszenerien mit dynamischen Hindernissen

5.3 Beispiel Testfall

Testfall	Beschreibung	Erwartung
TC01	Start -> Hindernis vor Ziel	Ziel wird auf alternativem Weg erreicht
TC02	Keine Hindernisse	Direkter Weg zum Ziel

6. Testkonzept und Qualitätssicherung

Zur Verifikation wurden JUnit-Tests entwickelt, um Kernfunktionen automatisiert zu prüfen.

• testPathFound(): prüft Pfadberechnung bei freiem Ziel
• testNoPathWhenBlocked(): testet korrektes Verhalten bei blockiertem Ziel
• PointTest: validiert Objektvergleiche und Hashing

Diese Tests stellen sicher, dass die Navigation robust und korrekt funktioniert.

7. Paketstruktur

• de.maehroboter.app – Einstiegspunkt (Main-Klasse)

• de.maehroboter.core – Logik (Point, GoalManager, Engine)

• de.maehroboter.util – Hilfsfunktionen (Logger)

• de.maehroboter.view – Darstellung der Umgebung (Map)

• de.maehroboter.test – Testklassen mit JUnit 5

8. Fazit

Das Projekt zeigt exemplarisch, wie sich iterative Softwareentwicklung mit agilen Methoden erfolgreich auf ein technisches Problem anwenden lässt. Alle definierten Anforderungen wurden erfüllt und die Traceability zwischen Spezifikation, Umsetzung und Test wurde konsequent eingehalten. Die Software ist modular erweiterbar und bildet eine solide Grundlage für weitere Funktionen wie Echtzeitsensorik oder grafische Interfaces.

9. Anhang

- vollständiges UML-Diagramm
- Traceability-Matrix
- Iterationsprotokolle