Universität Stuttgart
Institut für Automatisierungstechnik
und Softwaresysteme

Fachpraktikum Gruppe 3

Erstellung von Steuersoftware zur Navigierung eines Roboters durch einen Hinderniskurs unter simultaner Kartografierung des selbigen

Nahed Halouani
Tim Braun
Marcel Sauter
Waldemar Repp
Michael Streib

Prüfspezifikation

Prüfer:	Prof. Dr.-Ing. Dr. h. c. Michael Weyrich
Betreuer:	Dustin White
Tutor:	Hannes Vietz

Start: 24.12.2020 Abgabe: 12.01.2021

Dokument Versionsverwaltung

Version	Autor	QS	Datum	Status	Änderungen
0.1	WaRe		11.01.21	veraltet	Erstellung
1.0	NaHal	MiSt, MaSa	11.01.21	vorgelegt	Ergänzung von Prüffällen Formatierung

0 Inhaltsverzeichnis

1 Prüfanforderungen

Die allgemeinen Anforderungen einer Prüfung der Software ist wie folgt bestimmt:

- Prüfungen sind möglichst so zu gestalten, dass diese alle möglichen Arten von Werten (normale, grenzartig, fehlerhaft) abdecken dabei aber nicht die derzeitige Testart außer Acht lassen (Review, Simulation, Test).

- Prüfungen sind, nach den im Pflichtenheft definierten Umgebungen, zu treffen. Dies bedeutet insbesondere die Mindestabstände zwischen Hindernissen, die Funktion ohne Kartennutzung wie auch weitere im Pflichtenheft ausgewiesenen Grenzwerte sind zu beachten.

- Für die unterschiedlichen Prüfungen gilt besonders:
  - Reviews müssen, unter Beachtung der zuvor während dem Softwareentwurf und der Verfeinerung dessen getroffenen Entscheidungen, dem Styleguide sowie dem Ablauf und der Aufteilung, auf die sich das Team geeinigt hat, basieren.

  - Simulationen müssen die allgemeine Funktion unter Normalbedingungen testen, sind jedoch besonders für die Durchführung von vielen Tests in kurzer Zeit zu nutzen. Auch sind Simulationen für Tests von Grenzfällen zu nutzen, bevor dies als Vor-Ort-Test geschieht.

  - Tests sind, nach expliziter Angabe des Kunden, als Black-Box-Tests durchzuführen. Dies bedeutet auch dass es sich hierbei um Echtdaten handelt. Entsprechend sind Logs, Debug-Funktionen und Interaktion auszuschließen. Ferner sollten Daten nicht durch den Roboter aufgezeichnet werden, da nicht sichergestellt werden kann, dass nicht diese Daten Fehler aufweisen könnten. Eine Kalibrierung vor Ort ist jedoch explizit gewünscht, sofern diese nicht die Funktion auf einem anderen Roboter, der zwar dieselbe Hardware nutzt, jedoch leichte Abweichungen aufweist, beeinträchtigt.

- Es sind in allen Fällen möglichst diverse Hinderniskonstellationen zu testen und zu protokollieren.

- Prüfläufe müssen unter Beachtung der vom Kunden spezifizierten und im Pflichtenheft aufgenommenen Grenzen durchgeführt werden.

- Diese sehen unter anderem die Korrekte Funktion mit als auch ohne Karte vor.

- Prüfungen werden zunächst als Code Review, dann, sofern möglich, im Simulator und später vor Ort im IAS Testraum durchgeführt.

- Es muss, sofern möglich, ein möglichst breites Set von Eingaben, Konstellationen und Situationen abzudecken sein.

- Jeder Prüfungslauf muss dokumentiert werden und im Falle von fehlerhaftem Verhalten ein entsprechendes Dokument erzeugen (Non-compliance-report).

2 Methoden der Prüfung

Geprüft wird auf folgende Arten (nach Reihenfolge der Auflistung):

- **Statisch (Code review)**
  - Harte Fakten ((Logische) Zeilen an Code, Anzahl Methoden, Durchschnittliche Methodenlänge, etc.)
  - Kommentare bei GIT-Pull-Request (Pull blockiert weil...)
  - Methodik-Review bei Branch-Merge (Gutes/Schlechtes bei dieser Iteration)

- **Simulation**
  - Aufzeichnung von internem Softwareablauf durch Logs und Snapshots (relevante Systemattribute zu regelmäßigen Zeitpunkten)
  - Auswertung der erreichten Zeiten für Zielfahrt sowie Anzahl nicht erfolgreicher Fahrten.

- **Test (Vor Ort)**
  - Erfassung von relevanten Daten händisch, Auswertung durch Tabellen-kalkulation und anderen, üblichen Methoden

3 Prüfkriterien

- **Statisch**: Zunächst wird der Code auf mehrere Faktoren geprüft.
  - Korrektes Verhalten im Normalfall
  - Sinnvolles Verhalten im Grenzfall (Fehlerbehandlung)
  - Semantisch korrekte Struktur
  - Semantisch korrekte Informationsverarbeitung und -Äußerung
  - Codestyle
  - Leistung

- **Simulatorisch**:
  - Korrektes Verhalten im Normalfall
  - Sinnvolles Verhalten im Grenzfall (Fehlerbehandlung)
  - Deterministisches Verhalten (Nachvollziehbarkeit)
  - Leistung (kann im Simulator sinnvoller getestet werden als Vor-Ort, da dieser nicht in Echtzeit laufen muss).

- **Vor-Ort-Test**:
  - Korrektes Verhalten im Normalfall (zu Beginn)
  - Sinnvolles Verhalten im Grenzfall (zur Eliminierung von Fehlern, besonders relevant in Vor-Ort-Tests)
  - Deterministisches und sinnvolles Verhalten (Optimierungen können so besser erfasst werden da das Augenmerk jederzeit auf dem derzeitigen Lauf ist)
  - Leistung (Stoppuhr und Maßband erlauben Realwerte für Zeit pro Test)

4 Prüffälle

4.1 Hauptprüffall: Erfolgreiche Fahrt

- Wurde das Ziel erreicht?
- Ausgangssituation: Beliebige Position auf Spielfeld.
- Welche Eingaben: Alle zu Beginn anzugebenden Daten
- Ergebnis: Kurvenfahrt (Ja/Nein), Ziel erreicht (Ja/Nein), Zeit bis Ziel, Von Menschen verständliche Fahrweise (Ja/Nein)

4.2 Grenzfall: Minimaler Abstand zwischen Zwei Hindernissen

- Hat der Roboter den Spalt erfolgreich passiert?
- Ausgangssituation: Beliebige Position und Rotation vor Spalt
- Eingaben: Breite eines Hindernisses, Zielposition
- Ergebnis: Spalt passiert? Zeit bis Ziel, Sinnvolles Verhalten?

4.3 Grenzfall: Sackgasse

Der Roboter wird an einem beliebigen Punkt in einer Gasse gestartet, aus der er bis zum anderen Ende (dem Ende ohne Ziel) zu navigieren hat und dort einen Weg aus der Gasse zu finden hat. Von dort an muss der Roboter wieder umkehren, diesmal jedoch außerhalb der Gasse navigieren. Dies ist unter anderem ein Test, um sicherzustellen, dass der Roboter in ähnlichen Situationen auch wenn nötig zurück fährt zu einem früheren Standort.

- Hat der Roboter sich aus der Sackgasse befreien können?
- Ausgangssituation: Roboter in langer Sackgasse
- Eingaben: Zielposition
- Ergebnis: Sackgasse verlassen? Ziel erreicht/Sackgasse vermieden? Zeit? Sinnvolles Verhalten?

4.4 Hauptprüffall: Positionserfassung

Es ist zu prüfen, ob die erfasste Position des Roboters mit der tatsächlichen Position übereinstimmt.

- Ist die angezeigte Position nahe genug der echten Position?
- Ausgangssituation: Roboter an zufälliger Position auf Feld
- Eingaben: Keine
- Ergebnis: Welche Softwareposition entspricht welcher realen Position? Wie genau? Genauigkeit abhängig von Position? Andere Faktoren für Genauigkeit?

4.5 Hauptprüffall: Wegfindung

Es ist zu prüfen, ob die geplante Strategie den umgesetzten Aktionen des Roboters übereinstimmt.

- Verhält sich der Roboter der Strategie entsprechend korrekt?

- Ausgangssituation: Roboter erhält auffällige Strategie von der offensichtlich ist ob der Roboter dieser folgt.
- Eingaben: die zu nutzende Strategie
- Ergebnis: Folgt Strategie? Anomalien die nicht erwartet wurden?

4.6 Hauptprüffall: Kartographie

Es ist zu prüfen, ob der Roboter die Hinderniskonstellation kartografieren kann. Die Software soll eine Karte durch Nutzung der Infrarotabstandssensoren bei der Fahrt erstellen.

- Ausgangssituation: Beliebige Position auf Spielfeld.
- Eingaben: Keine
- Ergebnis: Können die Hindernisse vom Roboter erkannt werden? Kann die Karte erstellt und auf dem Rechner visualisiert werden? Ist die erstellte Karte konform zur Hinderniskonstellation? Wie genau ist die Karte?

4.7 Hauptprüffall: Erweiterte Wegfindung

Es ist zu prüfen, ob die Wegfindung mithilfe der Karte verbessert wird.

- Ausgangssituation: gleicher Ausgangspunkt wie die erste Fahrt.
- Eingaben: die zu nutzende Strategie, die Kartendaten.
- Ergebnis: Werden die kartografierten Hindernisse vermieden? Wird die Zeit im Vergleich zum ersten Durchlauf verbessert?

4.8 Prüffall bei fehlerhaften Werten: Falsche Benutzereingaben

Es ist zu prüfen, ob bei falscher Benutzereingaben auf der Benutzerschnittstelle die Software abstürzt.

- Ausgangssituation: Beliebige Roboter-Position, noch keine Eingaben auf der Benutzerschnittstelle.
- Eingaben: beliebige falsche Benutzereingaben
- Ergebnis: Zeigt die Software ein Fehler oder stürzt sie ab?

4.9 Visualisierung

Siehe Testfallspezifikation-Visualisierung