Universität Bremen

**Systeme hoher Sicherheit und Qualität**
**Fachbereich 03: Mathematik/Informatik**
**Prof. Dr. Christoph Lüth**
**Christian Liguda**
**Wintersemester 2013/2014**

# Übungsblatt 1 - Ergänzung

**Anmerkung:** Die einzelne Punkte werden in Bezug zu dem Beispiel des autonomen Fahrzeugs von Übungsblatt 1 bearbeitet. An einigen Stellen (z.B. Learnability) werden Beispiele für Techniken oder konkrete Anforderungen gegeben ohne das explizit auf das autonome Fahrzeug eingegangen wird. Da es hier jedoch um eine konkrete Anforderung geht (z.B. Videotutorials) wird auf eine explizite Erwähnung des Autos verzichtet, was in dieser Form auch auf den Übungszetteln in Ordnung war.

**Functional Suitability**
*Completeness:* Das Programm muss die auf dem Zettel angegebenen Aufgaben erfüllen. Dazu muss gezeigt werden, dass das System in entsprechenden Testszenarien wie gewünscht funktioniert, also Beispielsweise, dass es die Route zu einem gewünschten Ziel berechnen kann und beim Fahren bremst, wenn sich ein Hindernis ergibt dem es nicht ausweichen kann.
*Correctness:* Beispiel Hindernisserkennung: Es muss durch ausreichend viele Testszenarien sicher gestellt werden, dass zum einen Hindernisse erkannt werden und zum anderen, dass es nicht Hindernisse dort erkennt wo keine sind. Ferner kann für den Test spezifiziert werden, wie hoch der Anteil an *false positives* und *false negatives* sein darf.
*Appropriateness:* Da das System primär autonom handeln soll, ist dieser Punkt schwierig. Eine Unterstützung wäre bei der Routenplanung denkbar, wobei die Grenze zur *Operability* fließend ist. Die Unterstützung könnte durch eine Autovervollständigung bei der Zieleingabe erfolgen.

**Performance Efficiency**
*Time behaviour:* Die Reaktion auf ein erkanntes Hindernis soll innerhalb eines Zeitraumes t erfolgen (z.b. t = 10 ms).
*Resource utilization:* Besonders kritische Aufgaben (Hindernisserkennung/Bremssystem) benötigen ihre eigenen Ressourcen. Es ist darauf zu achten, dass einzelne Systeme nicht wiedersprüchliche Signale an externe Ressourcen (z.B. Bremssystem) senden. Hierfür wäre eine Angabe der Nachrichtenpriorität denkbar.
*Capacity:* Das autonome Fahren soll auch im Stadtverkehr möglich sein in welchem die Anzahl der Hindernisse sehr groß ist. Testbare Eckdaten wären: Empfohlenen Geschwindigkeit = 50 km/h, Anzahl der zusätzliche Autos in unmittelbarer Nähe, Anzahl der Verkehrsschilder auf der Teststrecke.

**Compatibility**
*Co-existence:* Die Software zur allgemeinen Steuerung soll auf einem Betriebssystem für Embedded Systems laufen auf welchem auch weitere Software installiert sein kann.
*Interoperability:* Um eine Kommunikation zwischen den Subsystemen zu ermöglichen oder um eine Kommunikation zwischen verschiedenen Fahrzeugen zu erlauben, wird ein spezielles Protokoll für die Kommunikation definiert, sowie ein XML Schema um die Daten der einzelnen Systeme in vorgegebener Struktur zu veröffentlichen oder zu verarbeiten.

**Usability**
*Appropriateness Recognizability:* Ein kurzes Demovideo der Software in Aktion gibt dem Kunden die Möglichkeit zu beurteilen, ob das System für ihn geeignet ist.
*Learnability:* Ein Benutzer soll die Software selbstständig erlernen können. Dafür können für jede Aufgabe kurze Videotutorials erstellt werden, welche bei Bedarf eingeblendet werden können. Alternativen wäre Hilfstexte bei den einzelnen Formularen um bei komplexen Aufgaben helfen.
*Operability:* Die meisten Menüpunkte lassen sich durch Icons direkt anwählen. Dabei sollen die Symbole eindeutig auf die Funktionalität schließen lassen. (Anmerkung: In der Übung wurde dieser Punkt unter *Appropriateness* genannt. Ich halte beide Punkte für schwer abzugrenzen. Unter *Appropriateness* könnte man jene Punkte zusammen fassen, die einen dabei unterstützen eine komplexe Aufgabe überhaupt richtig auszuführen. Zum Beispiel kann durch divide-and-conquer wird die komplexe Aufgabe in einfache Unterpunkte zerlegt werden oder durch eine flexible Schritt-für-Schritt Menüführung kann die Eingabe auf die wesentlichen Punkte reduziert werden (siehe Beispiel oben). Unter *Operability* könnte man eher „weichere“ Aspekte verstehen, wie sinnvolle farbliche Hervorhebung, Autovervollständigung oder Drag-and-Drop Möglichkeiten)
*User error protection:* Da das System primär autonom handeln soll, ist dieser Punkt schwierig.
*User interface aesthetics:* Ein Experte für die Gestaltung von User Interfaces sollte darauf achten, dass die einzelnen Menüfenster klar strukturiert und farblich harmonisch wirken.
*Accessibility:* Auch Menschen mit Sehbeeinträchtigungen sollen die Software bedienen können. Daher soll sie Schriftgröße verstellbar sein, sowie ein alternatives farbliches Layout Menschen mit einer Rot-Grün-Schwäche helfen. Wenn z.B. die Farbe grün eine schnelle Route hervorhebt und rot eine lange, kann dies für die betroffenen Menschen verwirrend sein.

**Reliability**
*Maturity:* Der normale Betrieb sollte ausreichend getestet worden sein. Dafür wird bei der hier entwickelten Software verlangt, dass sie sowohl in Simulationen als auch im Realbetrieb getestet wurde (z.B. in einer Simulation für x Wochen und im Realbetrieb für y Tage/Wochen).
*Availability:* Das System sollte nach Fahrbeginn immer zur Verfügung stehen. Dies kann durch ein parallel laufendes Zweitsystem sicher gestellt werden. Ferner sollten Updates welche die Funktionalität beeinträchtigen nur vor oder nach Fahrtantritt erfolgen.
*Fault tolerance:* Der Ausfall eines Untersystems kann für ein autonomes Fahrzeug kritisch sein. Eine Möglichkeit wäre, ein unabhängiges System zu haben, welches im Fall eines nicht erreichbaren kritischen Systems einen automatischen Bremsvorgang einleitet.
*Recoverability:* Der aktuelle Systemzustand (gewünschtes Ziel, geplante Route, aktuelle Position) sollte in kurzen Intervallen gespeichert werden. Sollte das Gesamtsystem abstürzen und neu

gestartet werden, so kann es anhand der gespeicherten Daten versuchen, den letzten Zustand wiederherzustellen.

**Security**
*Confidentiality:* Zunächst ermöglicht eine Benutzername/Pin Identifizierung nur autorisiertes Personal Zugang zu dem System. Über die Einteilung in mehrere Sicherheitsstufen kann auch nur Zugang zu einem Bereich gewährt werden.
*Integrity:* Mit Hilfe von Signaturen werden alle Datenbestände versehen. Dies kann auch für Protokolldateien gelten, damit diese nicht geändert werden.
*Non-Repudiation:* Sollte ein Eingreifen durch den Benutzer während der Fahr möglich sein, so sollten seine Anweisungen protokolliert werden. So kann im Unfallsfall nachgewiesen werden, ob der Benutzer oder das System Schuld am Unfall ist.
*Authenticity:* Jedes System, welches mit der Software kommuniziert, ist mit einem signierten Zertifikat ausgestattet. Dadurch kann jedes System eindeutig identifiziert werden.
*Accountability:* Da wird jedes System eindeutig identifizieren wollen (siehe Authenticity), soll ferner jede Anfrage eines Systems nur über eine entsprechen verschlüsselte Verbindung erfolgen, so dass zu jedem Zeitpunkt klar ist, dass die Verbindung zu dem ursprünglichen Teilnehmer existiert.

**Maintainability**
*Modularity:* Die Software wird in einzelnen unabhängigen Komponenten entwickelt welche über ein definiertes Protokoll miteinander kommunizieren. So kann jederzeit eine Komponente, z.B. zur Hindernisserkennung, ausgetauscht werden.
*Reusability:* Insbesondere die Module welche reine Berechnungsvorschriften enthalten, wie Hindernisserkennung, sollen völlig autark sein und nicht an andere Module gekoppelt werden, da sie in anderer Software benötigt werden.
*Analysability:* Der Einfluss einer Änderung in einem Modul auf ein anderes ist durch die Modularisierung gering gehalten und kann durch die Schnittstellenbeschreibung nachvollzogen werden. Zur Fehleranalyse soll ein Logggingframework verwendet werden, welches bei Bedarf unterschiedlich detailliert protokolliert (z.B. JLog). Der Grad der Protokollierung kann im Programm selbst geändert werden, aber auch extern um eine externe Fehlersuche zu ermöglichen ohne den Anwender mit einer Änderung des Logging Levels zu beauftragen.
*Modifiability:* Solange eine Komponente dem definierten Protokoll entspricht kann sie problemlos modifiziert werden.
*Testability:* Da eine Software zur Steuerung eines autonomen Fahrzeugs sehr robust sein muss wird erwartet, dass jede Komponente mit ausgiebigen Unit Test überprüft wird. Für besonders sicherheitsrelevante Aspekte, wie der Authentifizierung wird erwartet, dass die Sicherheit durch formale Tests bestätigt wird (Hier geht Testen schon in Verfikation über! Dies passt aber meiner Meinung nach, da es den Nachweis erbringt, dass die Komponente korrekt läuft.)

**Portability**
*Adaptability:* Jede Kommunikation zu anderen Komponenten, z.B. Datenbanksystemen, sollte über standardisierte Protokolle erfolgen um auch bei einer Änderung dieser Komponenten mit den anderen Komponenten weiter zu arbeiten. Sind keine standardisierten Protokolle vorhanden, muss eine sehr schmale Schnittstelle erstellt werden, welche die Anpassung der Daten an das spezielle

System vornimmt. Inwiefern sich Komponenten wie das Bremssystem an neue Systeme adaptieren lassen, hängt stark von der konkreten Ausprägung ab.
*Installability:* Hier nur bedingt sinnvoll, da die Installation von Spezialisten durchgeführt wird und nicht von einem normalen Softwarebenutzer.
*Replaceability:* Unsere Software ist einzigartig und wird nicht ersetzt :-). Da die Software keine große Datenverwaltungssoftware ist, ist dieser Punkt höchstens bei der Ersetzung von Teilsystemen sinnvoll. Hier sorgen standardisierte Protokolle und XML Schemata für eine Ersetzbarkeit von Teilsystemen.